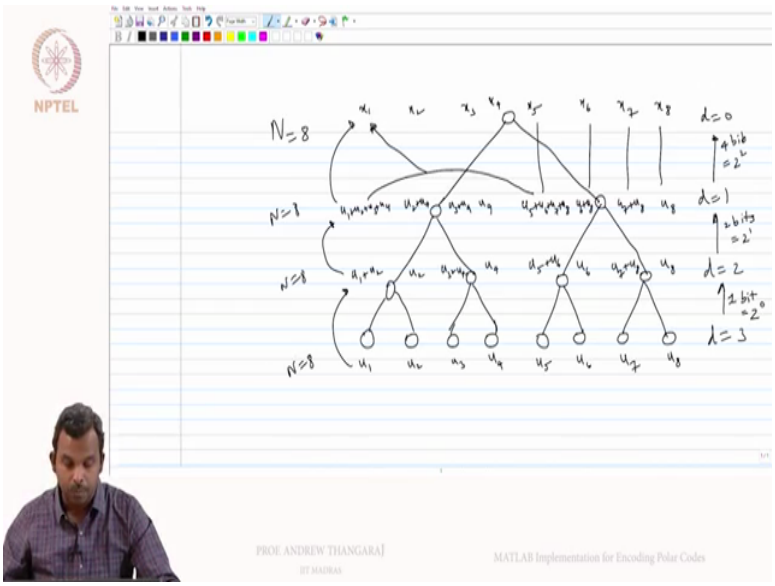
LDPC and Polar Codes in 5G Standard
Professor Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology, Madras
MATLAB Implementation for Encoding Polar Codes

(Refer Slide Time: 00:32)



Hello and welcome to this lecture on writing a MATLAB program for encoding the polar codes, we can start with brief description of how I will go about doing it and then start writing the code okay, so let me take an example of the, let us say the easiest example to take is the let us say that N equals 8 code, polar code so if N equals 8 if you think of the binary tree representation you have 8 libs and then 4 in the next level right.

And then 2 in the next level above that right and then finally the root node and the way encoding happens you have u1 u2 u3 u4 u5 u6 u7 u8, you need to go up all the way to the top, how do you do that and the first level you simply do this u1 plus u2 and then u2, so let me write it a bit differently here so that it is easy, so you have u1 plus u2 and then u3 plus u4 and then u5 plus u6 and then u7 plus u8, right.

So this is what happens up to the first level, after the second level you have u1 plus u2 plus u3 plus u4 right, basically then you have u2 plus u4 then you have u3 plus u4, u4 okay so two at a time you combine using the same logic then you have u5 plus u6 plus u7 plus u8, u6 plus u8, u7 plus u8, u8 then finally again you combine the two right so again you combine, I do not want to write the whole thing out and you will have x1, x2, x3, x4, x5, x6, x7, x8 these four are easy, these four are equal to this.

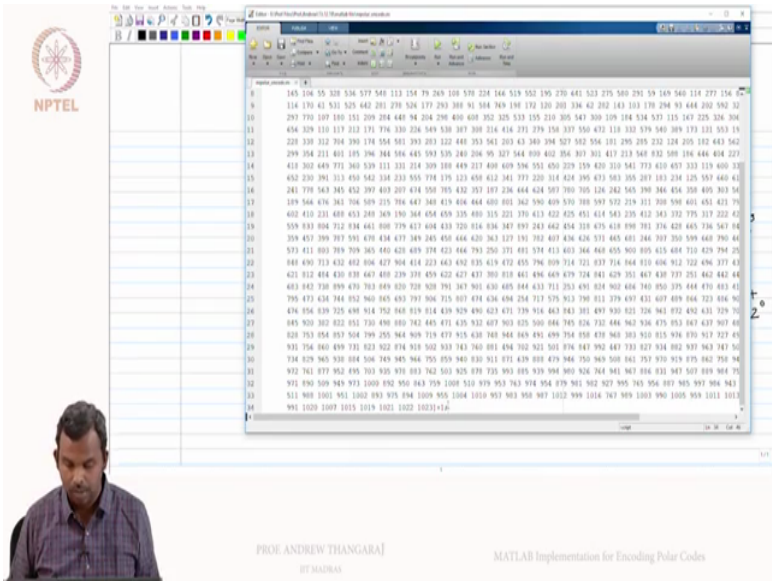
These four are the ex-or of this and this right so together goes here and like that likewise okay, so this is the logic and this is how we will implement it, we will initially assign the vector x to be equal to u itself and then sort of do in place computations as we go higher and higher so we go from here to here and then we will go from here to here and finally we will go from here to here.

Initially we will assign x to be this and we will process one layer at a time okay and the way I will write my code, I will write it in a general way so that it will work for a general depth, remember the depth goes as 0 here, 1 here, 2 here and 3 here, depth 3 you combine one bit at a time and here you combine, so one is 2 power 0, think of it as that, you combine here 2 bits which is 2 power 1 and then here you combine four bits right.

So four bits you combine here which is 2 power 2 okay so the number of bits you combine starts out with being 1 and then keeps getting multiplied by 2, in the next level you combine 2 bits at a time, next level you combine 4 bits at a time and so on okay but the total number of bits in every level is always 8 right so it is always 8 bits after every level okay except that you keep combining more and more at a time okay.

So that is the idea and we will write the code in this fashion so we will start with depth 3, so there will be a loop on the depths and for every depth there will be a number of bits that you will combine and then there will be a horizontal loop which goes through those bits and creates the next update okay so we will just keep the output vector is the input vector and keep doing in place to get to the final encoding okay.

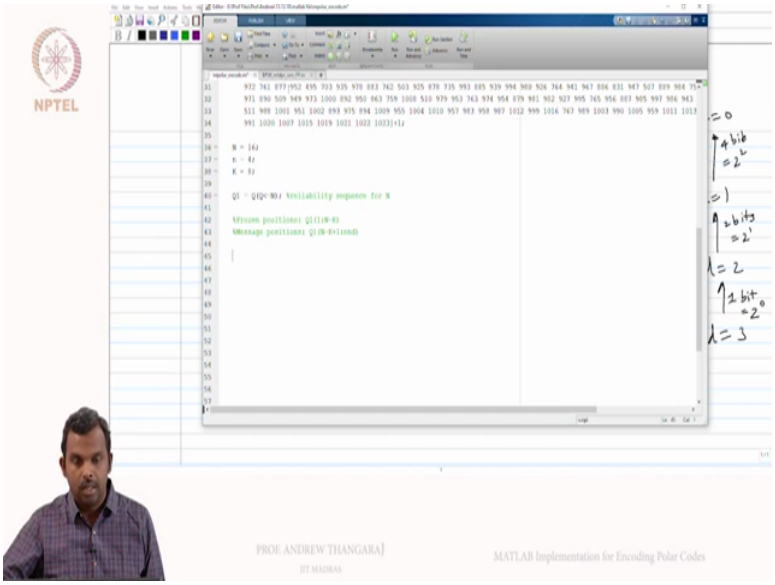
(Refer Slide Time: 4:43)



So this is the way I will write my code, let me get started okay so before that you need the reliability sequence and this is an array which is reliability sequence, it is available here so I have taken it from somewhere and put it here, this is the reliability sequence that I had you see it starts with 0 and ends with 1023, the standard specifies it from the with the indices 0 to 1023, of course you can add 1 to it to make it go from 1 to 1024.

We will do that right away okay so that this gets added, so that is the reliability sequence and then so now it is a question of adding the messages and let me give some room here so that you can see what I am typing, so one needs to generate messages and start doing similar operations to (())(05:31) so for that maybe I can open one of the, yeah so one of the previous things so that I do not mess up on simple things.

(Refer Slide Time: 05:50)



So okay so you need N and K, so let us pick n to be, I do not know, we will pick it as 16, just for starters and this gives small and equals 4 okay you can write code if you like to generate that but this is n equals 4 and then K, I need K maybe we will take as 8 okay, so now if you notice the reliability sequence goes from 1 to 1024 and what is the reliability sequence for 16 okay.

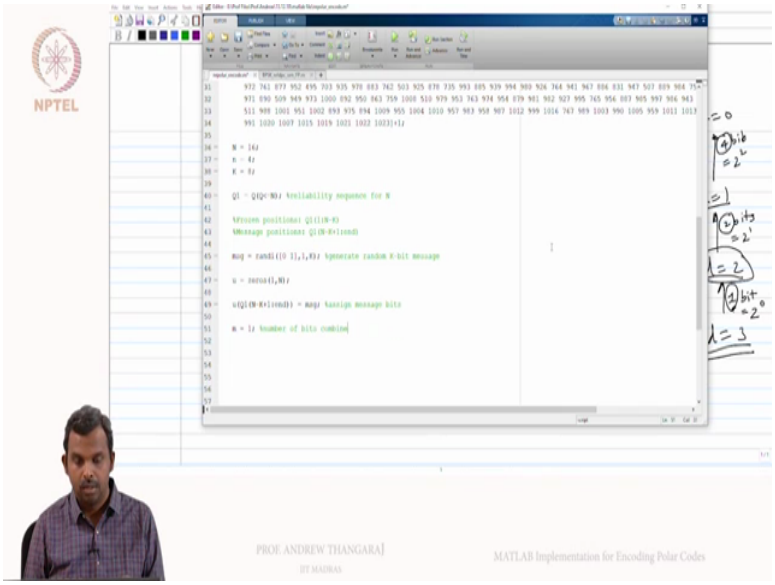
So turns out I will find Q1 which is the reliability sequence for this and the logic for that is quite simple, you take the original reliability sequence and take all the numbers that are less than or equal to 16 in it okay, so it is just Q of Q less than or equal to N okay, so hopefully this is clear to you, you look at all the values of Q which are less than 16 so you will start with 0, 1, 2, 4, 6, 16 is not there, 32 is not there, 3, 5 is there, 9 is there, 6 is there, 12 is there, 7 is there, 11 is there like that.

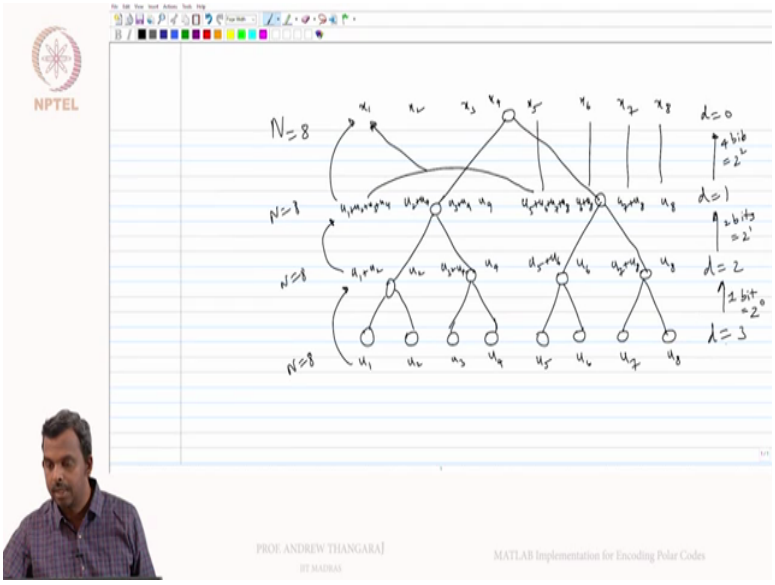
So in the same sequence 13 is there, 14 is there and then 15 comes somewhere and that will be the end okay, so that is the way the sequence will go so you start looking at this sequence and pick out only those numbers which are less than 16, so sort of efficient in some way, maybe not the most optimized but efficient in some way, so you get the Q1, Q1 is your reliability sequence okay.

So once you have the reliability sequence okay for the chosen N, you have a certain reliability sequence okay and then after that you are ready to find the frozen positions right, so frozen positions are basically Q1, 1 colon N minus K right, so the N minus K, the least reliable N minus K positions are frozen okay, so that I think okay and the message positions

are Q1, you could say N minus K plus 1 colon we can say end okay so it goes to the last okay, so that is the message positions.

(Refer Slide Time: 8:15)





So we are ready to generate the message, so let us generate a random message, I will just cut and paste from here so that this reduces efforts, should be capital K generate K bit message okay and then I am going to form this vector u which is at the base okay so how do you form the vector u, remember I am going to start doing in-place computations so I will call that the vector x okay.

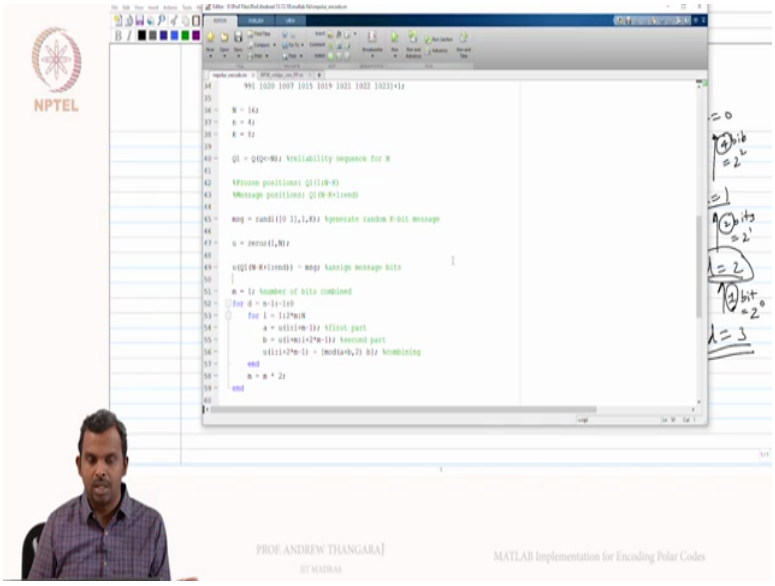
So what does my x, or maybe I will it u, I will initially make it 0, 1, N okay, so it is 0 and then the non-frozen positions, u of Q1 of N minus K plus 1, sorry minus K plus 1, colon N whatever is not to be frozen I have to let it be the message okay, so this is how it is, the

remaining are 0, the frozen are 0, you do not have to do anything more about it, this is the non-frozen positions, so this is assigning message bits okay so one can check these things later on if I made any mistakes or not.

The last K positions in the reliability sequence are assigned message bits okay so now we are ready to do the in-place computation for the code words okay, so let us do that, I will put a loop here, so if you remember here my loop starts with d equals 3 okay, my loop starts at d equals 3 and here I am combining 1 bit at a time okay, so after that I need to go to d equals 2, I will be combining 2 bits at a time and then d equals 1, I will combine 4 bits at a time and then I will be done, when I combine 4 bits at a time I am done right.

So it depends where you like to start with the d since the combination happens here, it is best to start with d equals 2 okay, hopefully see where I am doing this, so the combinations starts at this point right so you start working at depth 2, so we can start at d equals 2 and then the number of bits to combine is 1 okay so let me have some variables assigned for that.

(Refer Slide Time: 10:36)



So let us start with that, so I will say L, I will, l is a bit, M is the number of bits, I am going to combine that is 1, okay number of bits combined is good enough and then I need a loop, I will have a loop which is on depth and it will go from N minus 1 okay remember this is N equals 4 here, so N minus 1 down to 0 okay, I will end this loop here and I will think about how to start putting in there.

Okay so you have the N minus 1th loop and M is 1 and so now we can start combining 1 bit at a time so how do you combine the bits, so you can do I equals 1 colon okay so when I

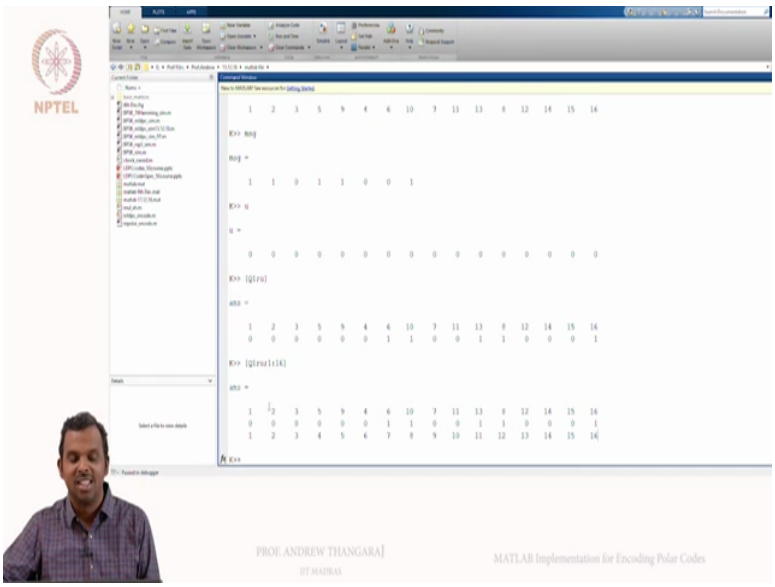
combine 1 bit I am going to combine u1 and u2 okay so when I is 1 I am combining I and I plus 1 and then my next jump is to I plus 2 right, so this needs to go as 2 into M colon N okay so this loop is for the combination okay.

So if I am combining m bits at a time I start with I, I go from I to I plus 1 minus M and then the next m bits are I plus M to I plus 2 M minus 1 okay and then I jump to I plus 2 N okay so those are the two things I have to combine okay hopefully the part is clear okay, so let us take the first part to be A which is u of I colon I plus M minus 1 okay and then B is the second part which is u of I plus M colon I plus 2 into M minus 1 okay.

So I have combined, this is the first part, this is the second part and now we are ready to combine, so u of I colon I plus 2 into M minus 1 is actually mod of A plus B, 2 and then B, is that right, so this is the operation okay so you can see what I am doing here, I go every M step and then I just simple combine the two to get the value for the next level okay, so once I am done with this I can multiply M by 2 okay.

So this is alright, so when you go from one depth to the next, the number of bits combined doubles okay so you combine 1 bit at a time at the lowest one, the next one you combine two bit at a time, finally by the time you are done you will be combining the whole thing more or less that will you the output u okay, hopefully this is clear, if there is no error then that should be it, that is it, I mean this is all that it takes to do encoding for polar codes, there is no major surprises here.

(Refer Slide Time: 14:22)



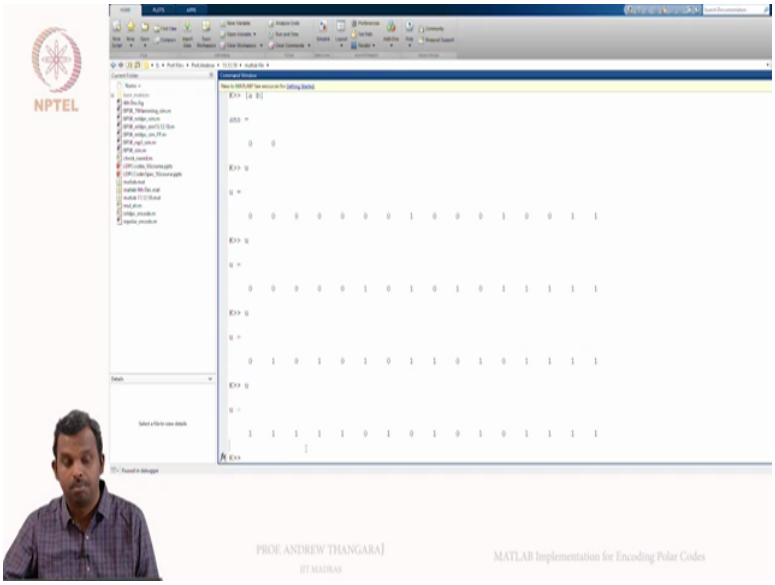
So maybe one needs to run this and see what happens okay so let me put a break point here, let me run it, change the folder, so you go up to message, so maybe it is good to see what the Q1 is okay so this is my frozen sequence 1, 2, 3, 5, 9, 4, 6, 10, 7, 8, 3 so on just to see, remember I am in debug mode so you have this k greater than greater than symbol as the prompt and Q1 is already assigned and then I generate my message okay so you can see what the message is okay so it is an 8 bit message 11011001 okay.

And then I generate my u, just to check u is the all 0 okay and then here it is, this is what is assigning u to the code word so if you see u now it is going to be, so I will do this I will put Q1 then I will put u so that you can see right below Q1 where you got a sign so if you see the 16th position has gone to 1, okay so how do you read this, so you should look at, so the first 8 has to be frozen, so 1 has to be frozen let us read like that, 1 has to be 0, 2 has to be 0, 3 has to be 0, 5 has to be 0, 4 has to be 0, 9 has to be 0.

So 9 comes right here, so 9 is 0 okay and then 10 has to be 0, this is 10 okay, 9 and 10 are 0 and then 1, 2, 3, 4, 5 is 0 okay, 6 is also 0, okay so 1, 2, 3, 4, 5, 6 and 9, 10 are 0 okay, what about the remaining positions if you start with the remaining positions, you start with 7, 7 and 11 have to be 1 okay so 7 is 1, 2, 3, 4, I think reading it with Q is a bit confusing so maybe I will also do 1 colon 16, so that we see the actual order okay.

So it is good to see the actual order also so you see 1, 2, 3, 4, 5, 6 and 9 and 10 are zeroes okay so they have been frozen you get 0 and then you need 7, 11, 13, 8 to be 1101, 7, 11, 13, 8 is 1101 okay so you can see 7, 11, 13 and 8 okay that is 1101 and then you need 12, 14, 15, 16 to be 1001 so you see 12, 14, 15, 16 as 1001 so the message bits have been assigned correctly, it was a simple piece of code but to check it is a bit confusing because these permutations are always bit confusing.

(Refer Slide Time: 17:40)



So you put the permutated sequence there and the original index here to see how the match happens so the assignment has happened correctly, there is no problem there so now that we have the correct thing so let us start looking at this okay, so you got this, you got this okay and then let us check what A and B are okay, so A and B should both be 0 okay and u should just not get altered at the end of this okay, so what I will do is I will run up to this point and finally check okay so that the first step is fully over and then you can see u okay.

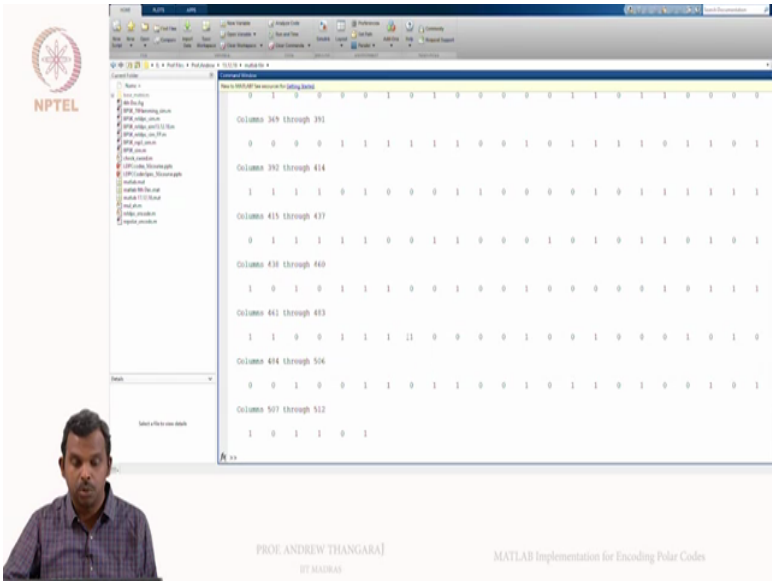
The original u was here okay and you can see how the processing has happened 00 went to 00, 00 went to 00, 00 went to 00, 11 became 01 okay so we know 11 has to become 01, 00, 00 again 11 became 01, 00, 00, 01 will become 11 okay so the first step has happened very correctly, there is no problem there so let us run to the cursor once again so that it finishes the whole of the second loop.

And let us check if the second thing got done correctly or not, now let us look at u, 0000 should go to 0000 right so that is the combination that is okay, 0001 should go to 0101 that happened correctly 0001 again should go to 0101, 0011 should go to 1111 okay, so, so far so good, that has happened correctly and then let us run to the cursor once again, this time it will be combining 4 bits at a time.

Okay so previously it was 2 bits at a time, it will combine 4 bits at a time, if you look at 4 bits these are the 8 bits here okay, 0000 0101 and then it is just going to become 0101 0101 okay that is correct and then the next 4 bits at a time is these guys 0101 1111 is going to become 1010 1111 that is also fine and now we are up to the last step, now if you run to the cursor

you should get the last step and that should be just the overall combination okay so overall combination is 8 at a time, this and this.

(Refer Slide Time: 20:16)



So if you ex-or, you are going to get 1111 1010 1010 1111 perfectly fine, so if everything went fine if I continue it should just end it ended okay so the u you have is the encoded version the message is the original message okay, that is the simple encoding for polar codes okay, so you can change N to anything else so for instance I can make N as 512 just for fun, the small n will actually be 9 so one of the things to do is to simply let it be log 2 of n, so that you do not have to keep changing this all the time.

K is maybe I can write this down below so that and n k k could be I do not know 300 you can do this, so if you do this if you encode you will generate one message and you will encode okay so it is out of heart to check when you run it try it and it will give you answer okay, so you can see the message if you like, it is a huge 300 bit sequence and then you can see the codeword u which is the 512 bit sequence okay so this works quite fast, it is a pretty neat little code, it is not, there is not major issues in writing this code okay.

So you can change, fool around with this numbers little bit and change it around, a few warning once again about the 5G standard, if you read the 5G standard, there are three steps in the polar encoding okay, the first step is you take the message bits in and then you append a CRC okay, CRC is something I never spoke about in this class, even for LDPC codes it is there, there is a CRC addition, CRC cyclic redundancy check, some addition bits about 10 bit, 11 bits or 6 bits or 24 bits or so many other bits that you add to the message for some

final protection and for the higher layers to find out if the decoded code word is correct or not okay for that purpose there is an addition of some CRC bits okay so that happens and after the CRC bits are added the actual polar encoding happens, so this step here will have a CRC addition before it.

So if I put K equals 300 there will be somebody called A which is the number of message bits that are coming in the standards I think it is also called A and then L will be the length of the CRC, I think it is called L in the standard document as well and then K becomes A plus L okay so you get the overall total number of bits that way okay and then Q and Q1 and all of that is the same, you find the U okay finding the U is exactly like this okay and then after this there is a rate matching step okay.

There are few more bells and whistles in the standard for instance there are some block interleavings in the middle, some other type of interleaving in the middle and some parity bit additions in some few of the cases I believe in one of the directions either uplink or downlink for a few short block length there is some parity addition, the interleaving of the CRC bits is done also for some reason so lot of intermediate steps in the standard, I am skipping all of that as far as this concerned.

To us from the coding point of view this is the most critical step, all the other things come and little bit of additions okay interleaving and parity check additions are also parity check bits are also related to the coding aspect but we are not focusing on that in the class so later on I will comment on the role they play and point you some references on how to use them if you like it okay.

But this is the core of the polar encoder which is the main part of the course, this is the (()) (23:25) so this is how you write the encoding okay, so hopefully it was clear to you, it is a simple enough encoding okay so this is the end of this lecture I will welcome you to look at this code it is available for you to download, you can go play around with it, change some parameters, maybe try and make it more efficient okay, can you identify one place where it can be made more efficient, I think the inner loop can definitely be saved, you can write one loop free command in MATLAB for that but maybe the outer loop is a bit more difficult to modify okay alright thank you very much we will meet again in the next lecture where we will start talking about decoders for polar codes.