LDPC and Polar Codes in 5G Standard

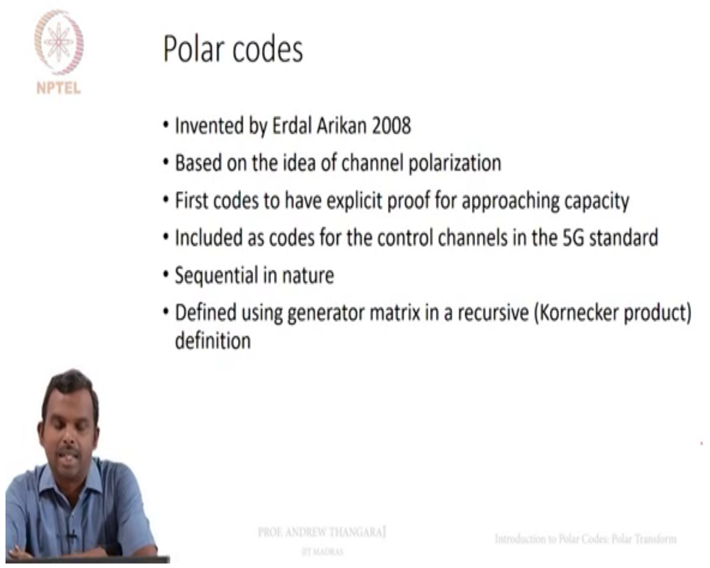
Professor Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology, Madras

Introduction to Polar Codes: Polar Transformation

Hello, welcome to this lecture from this lecture onwards we will start looking at polar codes. Polar codes as it turns out are very recent that were proposed in around 2008 by Erdal Arikan that is become very famous very quickly and it has very celebrated properties it is one it is a very nice code it has a very simple structure, very elegant and very nice performing good performance with very easy to describe decoder, in many ways it is sort of it has some of the properties of low density parity check codes as in the decoder is soft in soft out sort of decoder.

You can build a soft in soft out decoder and then there is a very simple structure this is sub-structure of parity checks and repetition which are combined in a very intelligent and smart way to create this and all credit goes to Erdal Arikan for this definition and construction of polar codes.

(Refer Slide Time: 01:22)



Polar codes

- Invented by Erdal Arikan 2008
- Based on the idea of channel polarization
- First codes to have explicit proof for approaching capacity
- Included as codes for the control channels in the 5G standard
- Sequential in nature
- Defined using generator matrix in a recursive (Kornecker product) definition

So let us get started, like I mentioned polar codes were invented by Erdal Arikan in 2008 the basic idea is one of what he introduced this, this idea of channel polarization, how you can take a noisy channel and through clever techniques create ideal noiseless channels and extremely noisy channels out of a single noisy channel. So this idea of combining multiple instances of the channel cleverly to get either noiseless ones or very noisy ones, more noisy ones than the original channel is this channel polarization.

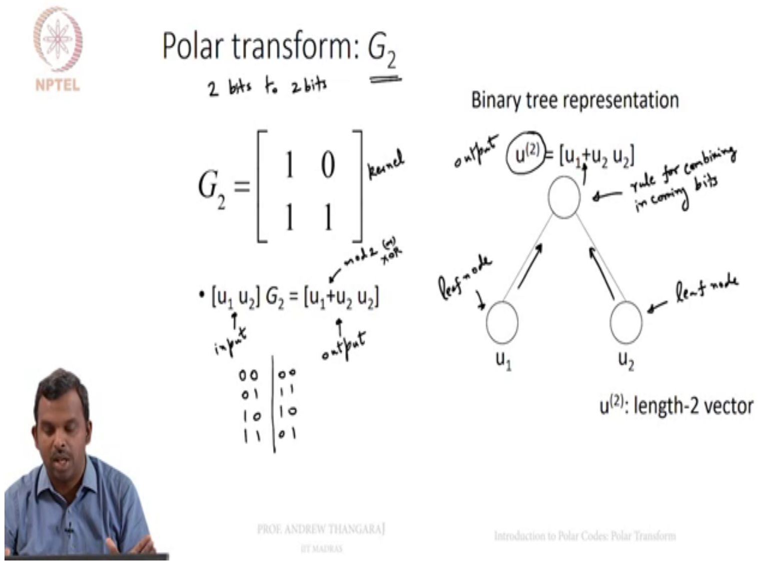
It is sort of like no law of averages you can't change the average some sort of energy conservation if you like, the channel is has a certain noise level you can't change that but if you do enough repetitions some of them become noiseless, some of them become very noisy so this is sort of like capacity it has a very nice information theoretic structure and wonderful way in which it can be developed from an information theory point of view, unfortunately we won't do such development in this lectures.

Our lectures are focused on implementation, so we will talk about them in a very sort of direct way focusing on how to get them implemented, how to implement the encoder? How to implement a decoder? How to get good performance from that, so that will be our focus ok. So a few points to know note is that this is a first codes to have explicit proofs of approaching capacity ok. So that's a very big selling point, these are included for as codes for the control channels in the 5G standard, the data channels have LDPC codes we saw them earlier and the control channels have polar codes in the 5G standard.

I wanted the big differences between LDPC codes and polar codes is the decoder here is sort of sequential. You decode the first bit and then the second bit in a certain order we decode the first bit, second bit, third bit so on ok so that's how you do it. So the decoder is slightly different then we have to think about it differently but interestingly the ingredients are the same they are single parity check decoder that we had therefore so single parity check decoder the Tanh rule and the mean sum, the mean sum will play an important role again and also the repetition idea, ok so this are the two key ideas and this decoder as well but except that the scheduling of the operations is very different and is specific and sequential ok.

And also the definition is not primarily through a parity check matrix you could do a parity check matrix as well but this is not primarily dome through a parity check matrix it is done through a generator matrix and in fact this is interesting polar transform which controls how the definition is done we will study that in detail ok, so let us get into it without any further interaction ok.

(Refer Slide Time: 04:09)



Polar transform: $\underline{\underline{G_2}}$

2 bits to 2 bits

$$G_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \text{kernel}$$

- $\underset{\text{input}}{[u_1\ u_2]}\ G_2 = \underset{\text{output}}{[u_1 + u_2\ u_2]}$ ← mod 2 (or) XOR

00	00
01	11
10	10
11	01

Binary tree representation

output $\boxed{u^{(2)}} = [u_1 + u_2\ u_2]$

$u^{(2)}$: length-2 vector

PROF. ANDREW THANGARAJ
IIT MADRAS

Introduction to Polar Codes: Polar Transform

So the most basic construct in polar code is the so called polar transform or Arikan transform you can call it, it is a very-very smart idea. It has roots in information theory and you can wonderfully describe it in your information theoretic terms but we will use a simple operational definition to describe what it is ok. So this is the main polar transform, it is a transform which takes 2 bits to 2 bits, it takes 2 bits to 2 bits, ok so this is the polar transform G2 there are also longer versions of it as G4, G8 for any power of 2 you have a polar transform but let start with the basic starting point which is taking 2 bits to 2 bits and the transform is very easily define it is defined by this kernel or matrix you can all it the polarization kernel or transform kernel is just 1 0 1 1 that's it ok.

You have u1, u2 this is the input to the transform ok the output is u1 u2 multiplied by G2 and this is the output ok, remember this plus id modulo 2 addition or ex-or you can think of it as binary ex-or ok so if you want you can make a little table it is quite easy to write down the table for this, the input is 0 0 the transform is 0 0 if it is 0 1 the transform is 1 1, it is 1 0 the transform is 1 0 and if it is 1 1 the transform is 0 1 ok. So it is an invertible transform it is quite a simple transform is nothing more to talk about it but you will see the way you can use this to cleverly combine and create transforms for longest strings and then what properties they have that is crucial ok, so you will see it works out very nicely ok.

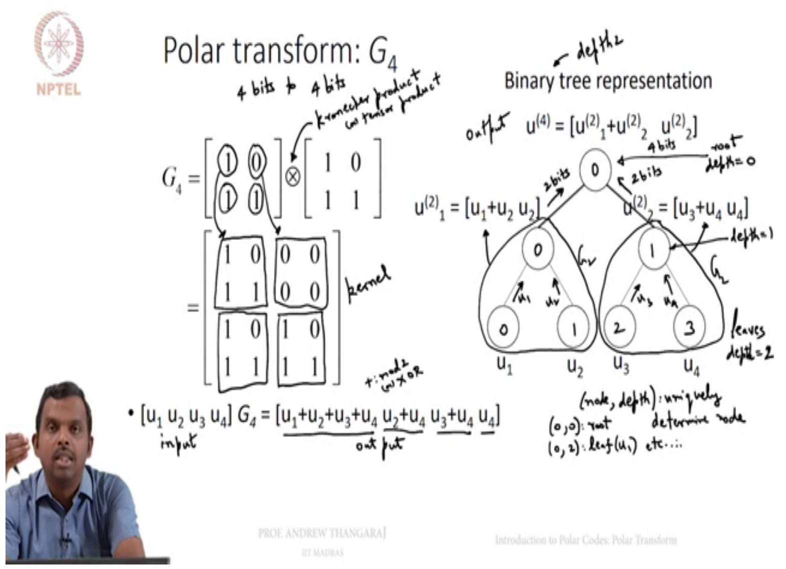
So all big stories have a very small staring point so this 2 by 2 transform is the starting point for polar codes right. So there is binary tree representation for this transform at this point it really sounds quite redundant and complicated to draw graphs for this 2 by 2 transform but it turns out as we compose more and more of this together and get larger transforms this binary tree picture is very important plays an important role in encoding and decoding and in throughout polar codes this binary tree representation play an important role much like the (())(06:39) graph played a role in the LDPC decoders this will play an important role for polar codes.

Ok so what is this representation? You have your inputs at the leaf nodes this is the leaf nodes of the tree this is also a leaf node, so this is sort of output node of this simple tree you have the input bit one u1 here another input bit u2 here and then the output which I am calling as this u2 ok this is just this 2 denotes that it is a length 2 vector we will see later on we will have length 2, length 4 vectors and we will combine them I the same way so this notation sort of important to know to so if I say u2 it means there is just length 2 binary vector and in this case it is just the output.

So we have u1 coming in from here, u2 coming in here these two are process that this node and outcomes u1 plus u2 and u2 ok so this is the transform ok. So is nothing much describe here it is very simple representation for what is going on here. So instead of thinking of the u1 and u2 as bits multiplying the matrix on the left we think of u1 and u2 as coming from the leaves of the tree and they come up the tree and they get processed to the next node and outcomes the output u1 plus u2, u2 ok

So remember this again whenever you go up a tree you can do u1 plus u2 and then u2 retain u2 as it is and you do u1 plus u2 ok so this is a rule that we will remember so this is a combination rule, rule for combining incoming bits ok, so whenever bits come in what you do at the node? You take u1 plus u2 and u2 so it is simple transform you see this are composed together in a clever fashion to get larger transforms. Ok so this is the basic transform make sure you have a good hang of it because as you go further down the slide this will be used in more and more interesting ways ok.

(Refer Slide Time: 08:53)



Polar transform: G_4

$$G_4 = \begin{bmatrix}1&0\\1&1\end{bmatrix}\otimes\begin{bmatrix}1&0\\1&1\end{bmatrix} = \begin{bmatrix}1&0&0&0\\1&1&0&0\\1&0&1&0\\1&1&1&1\end{bmatrix}$$

- $[u_1\ u_2\ u_3\ u_4]\,G_4 = [u_1+u_2+u_3+u_4\ \ u_2+u_4\ \ u_3+u_4\ \ u_4]$

Binary tree representation

$u^{(4)} = [u^{(2)}_1+u^{(2)}_2\ \ u^{(2)}_2]$

$u^{(2)}_1 = [u_1+u_2\ u_2]$

$u^{(2)}_2 = [u_3+u_4\ u_4]$

4 bits to 4 bits
kronecker product (or tensor product)
kernel
input
output
+: mod 2 (or XOR)
depth 2
output
4 bits
root depth=0
2 bits
2 bits
depth=1
G_2
G_2
leaves depth=2
(node, depth): uniquely determine node
(0,0): root
(0,2): leaf(u_1) etc...
PROF. ANDREW THANGARAJ IIT MADRAS
Introduction to Polar Codes: Polar Transform

So this is G4 ok, so I said G2 was a 2 by 2 transform, G4 is going to be a 4 bits to 4 bits transform ok it takes inputs 4 bits and puts out output 4 bits and its kernel is time using what is called a kronecker product so also called tense product of matrices sometimes kronecker product is very-very popular how do you do the kronecker product? If you have kronecker product of two matrices a and b what you do is you replace each entry of the first matrix by that entry times the second matrix ok, you replace one with one times the second matrix ok which is what you did here and the second entry is zero gets replaced by zero times the second matrix which is what has happened here.

Ok and likewise for the third entry and the fourth entry ok the third entry got replaced by third entry times the second matrix and fourth entry got replaced by fourth entry times the second matrix ok. So from two 2 by 2 matrices we end up getting a 4 by 4 matrix and that becomes the kernel for G4 which is the 4 bit polar transform or 4 bit Arikan transform ok. So this is the key idea here so you compose or you product out kronecker product out two G2's to get G4 ok.

So now if you multiply that 4 bit input here u1, u2, u3, u4 with G4 this is the output you going to get ok so it is easy to see this one u1 plus u2 plus u3 plus u4 is the first one u2 plus u4 is the second one u3 plus u4 is third one and u4 itself is the fourth one ok once again remember plus is modulo 2 or ex-or right binary ex-or or modulo 2 ok. So any input will give you the output and it is also invertible you can prove this in multiple ways, it is an invertible transform ok so this is the 4 bit polar transform ok.

So this 4 bit polar transform also has a tree representation ok so it is sort of build in the same way you have so you remember this is like first G2 this is G2 this is another G2 right so you have u3, u4 and u1, u2 going through two G2's ok you get two vectors of length 2 so this is the output of G2 right so this is G2 gives you this output this G2 gives you this output right, u3 plus u4, u4 whenever two bits come in right u1 comes in here u2 comes in here what you do in the combination? Rule for combination is? U1 plus u2, u2 same thing happens here when this 3 bits this 2 bits come in u3 and u4 is simply be u3 plus u4, u4 ok.

And then what you do with the next level so when you go from G2 to G4 you take two such things and then combine them further again ok, how do you combine? The same rule ok, so this goes on this edge this u2 to goes on this edge and what you do to get u4? This is the output you combine them again, how do you combine them? Two sets of bits are coming in now this are not single bits anymore this are two bits ok so this are two bits two bits remember this is one bit one bit down here in G2 you got two bits out.

If you have two bits two bits coming in you will get 4 bits out, how will get the 4 bits? You take this two incoming bits you add them keep them as first part and the second one alone (())(12:52) as the second part ok the same rule that you use, you use over and over again you will get the output, is that ok. So previously we saw that the G2 was a very simple and direct notation here G4 is a little bit more complicated but it is the exact same thing as what this multiplication by G4 achieves ok except that you have a binary tree representation you have u1, u2, u3, u4 and the leaves ok this are the leaves, leaf ok and then you have two levels in the binary tree.

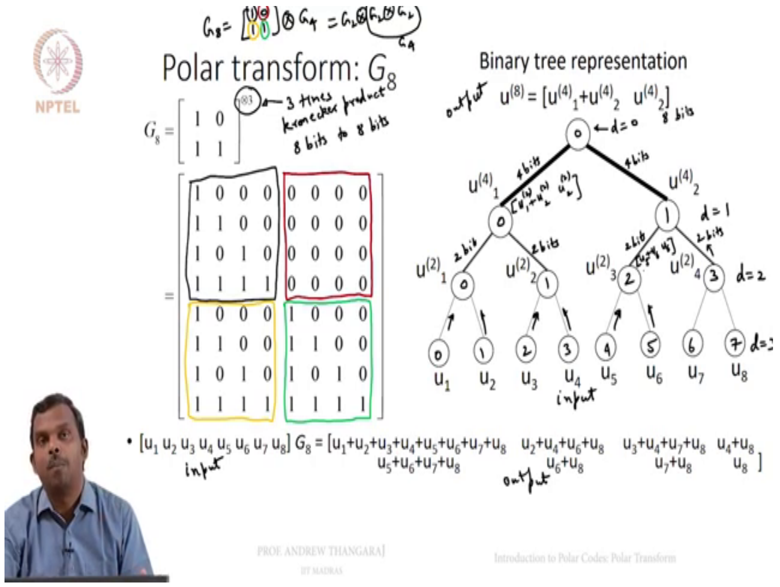
So this is suppose to have depth 2 ok so this has depth 2 here the previous one had depth 1 ok so this had depth 2, this is the root people call this the root of the tree this is suppose to have a depth of 0 ok this has a depth of 1 this level is depth 1 and this level leaves are at depth 3, depth equal 2 sorry this are terminologies we will use for this tree ok we have a root node which is at that zero and then below that root node there are two children this are said to be children of that root node ok at depth 1 and below every child there are two more children ok.

So each node at depth 2 there are two nodes at depth 2 and then below that you have two more children ok so we will also number this nodes this will call node 0, this I will call node 0 node 1, this I will call node 0, node1, node2, node 3 ok so remember how I am numbering the nodes I am just starting with zero on the left and going through 0, 1 etc 0, 1, 2, 3 ok so now node number and depth together uniquely depend in the node ok that is important. So for instance 0, 0 is the root 0, 2 is a leaf etc leaf and left most side leaf that has u1 etc ok.

So this is something important to know as well so this binary tree representation represents the multiplication by the polar transform to get the 4 bits out, the 4 input bits are at the leaves of the tree ok they are combine two at a time they go to the next level one higher level and then there again they are combine there you will get two bits at each node they are combined further to get 4 bits ok. The combination the rule for the combination is always the same, 2 bits are incoming you ex-or them keep them as the first part and then take the second one alone and keep it as a second part ok, so that is what you do in every node as you go up the tree ok.

So if you do that you get the transform, this is the binary tree representation for G4 ok. So you can imagine what G8 will be ok so we have G2 then from G2 we went to G4 which is G2 kronecker product with G2 then you will go to G8 ok so that is the next step.

(Refer Slide Time: 16:02)



$G_8 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \otimes G_4 = G_2 \otimes \underbrace{(G_2 \otimes G_2)}_{G_4}$

Polar transform: G_8

$$G_8 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes 3} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&0&0&0&1&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \\ 1&1&1&1&1&1&1&1 \end{bmatrix}$$

3 times kronecker product
8 bits to 8 bits

- $\underbrace{[u_1\, u_2\, u_3\, u_4\, u_5\, u_6\, u_7\, u_8]}_{\text{input}} G_8 = \underbrace{[u_1+u_2+u_3+u_4+u_5+u_6+u_7+u_8 \quad u_5+u_6+u_7+u_8 \quad u_2+u_4+u_6+u_8 \quad u_6+u_8 \quad u_3+u_4+u_7+u_8 \quad u_7+u_8 \quad u_4+u_8 \quad u_8]}_{\text{output}}$

Binary tree representation
output
$u^{(8)} = [u^{(4)}_1+u^{(4)}_2 \;\; u^{(4)}_2]$
8 bits
d=0
4 bits
4 bits
$u^{(4)}_1$
$u^{(4)}_2$
$[u^{(2)}_1+u^{(2)}_2 \;\; u^{(2)}_2]$
d=1
2 bits
2 bits
2 bits
2 bits
$u^{(2)}_1$
$u^{(2)}_2$
$u^{(2)}_3$
$u^{(2)}_4$
d=2
0 1 2 3 4 5 6 7
u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8
input
PROF. ANDREW THANGARAJ
IIT MADRAS
Introduction to Polar Codes: Polar Transform

G8 is the kronecker product three times ok, so you will get a 8 by 8 matrix so G8 will take 8 bits to 8 bits ok, so how do you do kronecker product three times? You already did kronecker product two times before G4 right, so you can also write G8 as G2 1 0 1 1 kronecker product with G4 ok. So you take G4 how would you do 1 0 1 1 kronecker product with G4? You replace the first element with G4 itself ok, so you will see this as G4, you have G4 here and you replace the second element okay maybe I can do some different colors here.

You replace the second element with zero times G4 ok and then you replace the third element (maybe I pick a different color) you replace the third element with one times G4 ok that is how you did a kronecker product you multiply replace each element by that element times G4 and fourth one we pick a blue green color that is the fourth one ok. So that got replaced by one times G4 so when you think of G8 you can think of it as G2 kronecker product G2 kronecker product G2 ok it is over that it is also equal to like I said kronecker product three times you do this first you get G4 so it is the same as G2 kronecker product G4 and how do you do kronecker product?

You replace each element by G4 that element times G4 so one times G4 zero times G4, one times G4, one times G4 that gives you G8 ok, now this you can repeat you can do G2 kronecker product G8 you will get G16 similar G32, G64, G128, 256, 512, 1024 what to 1024 is actually used in the 5G standard ok. So this is how the G8 looks and you can take 8 inputs bits and we will multiply the G8 and will get the8 output bits ok and you can also have a binary tree representation ok.

So here you have the root at depth 0 I will just write D for depth, depth 0 node 0 you have node 0, node 1 this is at depth 1 then you have node 0, node 1, node 2, node 3 is at depth 2 then you have node 0, node 1, node 2, node 3, node 4, node 5, node 6, node 7 this is at depth 3 ok so you have a tree of depth 3 ok and binary tree fully full binary tree complete binary tree (())(19:19) ok every node has two children ok all the way upto the depth of 3.

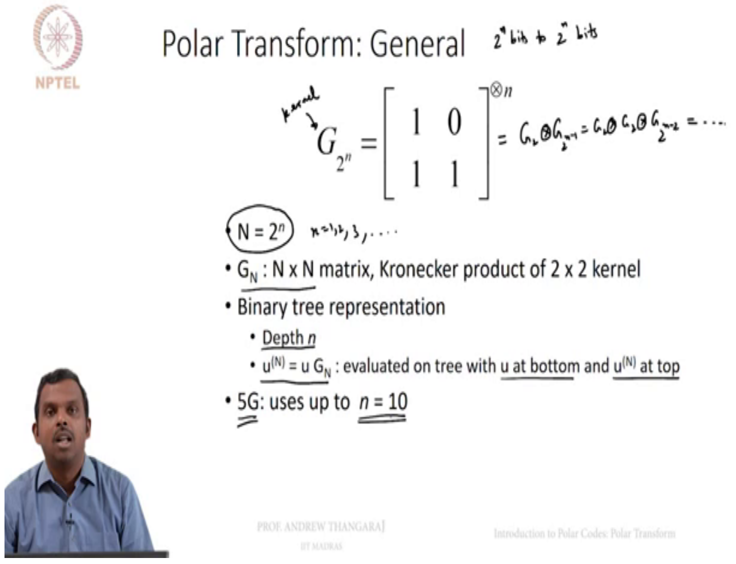
At the leaves you put the input ok and then you start processing anytime you hit a node the processing is very simple ok two one bits coming you get a 2 bit output ok so you have 2 bits here ofcourse you have 2 bits here what will this 2 bits be for instance if you want to write down this 2 bits it will be u5 plus u6 and then u6 right u5 and u6 are coming in u5 plus u6 and u6 ok that is the way you comeback. In here again 2 bits this are coming in and what will this guy be? This will be actually 4 bits ok how will you combine these two? This will be just u21 plus u22 and then u22 right for instance.

Ok this one will also be u23 plus u24 and then u24 by itself that will be this 4 bits and this 4 bits are combined again to get 8 bits this is the output and that is 8 bits right, so this is just the exact same thing as multiplication by its matrix and has a binary tree representation ok. So this polar transform for any n could be G2 or G4 or G8 or G16 you can see how you can do G16 right you take two such things you have this two 8 bits coming in you combine them you get 16 bits out ok.

So you have 16 leaves at the bottom and it will be a depth 4 tree and you will have a binary tree representation. Ok so this tree representation is very important to visualize to see how the polar code works and the decoder works. So far I have not defined the polar code once again be very clear, this is just the polar transform this is just a transform from 8 bits to 8 bits ok. It takes 8 bits input puts out 8 bits ok. The polar code is constructed using this transform in a very clever way so let us see how that is done in the next few slides.

But hopefully this transform is clear to you, it is very-very crucial that you have a good idea of what the transform is. You can take a few examples and run through this and see how it works you can implement this transform in MATLAB we can do that so that we can do encoding of polar codes ok so this is something very important to so alright ok.

(Refer Slide Time: 21:41)



Polar Transform: General 2^n bits to 2^n bits

$$\overset{\text{kernel}}{G_{2^n}} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n} = G_2 \otimes G_{2^{n-1}} = G_2 \otimes G_2 \otimes G_{2^{n-2}} = \dots$$

- $\boxed{N = 2^n}$ $n = 1, 2, 3, \dots$
- $\underline{G_N : N \times N}$ matrix, Kronecker product of 2 x 2 kernel
- Binary tree representation
  - $\underline{\text{Depth } n}$
  - $\underline{u^{(N)} = u\, G_N}$: evaluated on tree with $\underline{\text{u at bottom}}$ and $\underline{u^{(N)} \text{ at top}}$
- $\underline{\underline{\text{5G}}}$: uses up to $\underline{\underline{n = 10}}$

So what is the general picture? Ok so the general picture is you can have a polar transform for N equals 2 power n ok and this small n is 1, 2, 3 so on ok any number you pick. What is the kernel of this transform? The kernel is G2 n this is the kernel it has 2 power n bits to 2 power n bits inputs is 2 power n bits output is also 2 power n bits and the kernel is the kronecker product of 1 0 1 1 kronecker product n times ok. So you do it n times, so how do you actually write the down so you can think of it as G2 kronecker product G2 power n minus 1 which is G2 kronecker product G2 kronecker product G2 power n minus 2 and so on. Ok so finally if you want you can write it as G2 kronecker product G2 kronecker product with itself n times ok.

So this is the where the kernel is defined so this G n is n cross n matrix the binary tree representation will have depth m, one root and 2 power n leaves and the leaves will be of depth n and this is the multiplication ok. So it is evaluated on the tree with u at the bottom and u n at the top and 5G like I said uses up to n equals 10 ok, so this is the polar transform. This general picture hopefully this was clear to you and that's this is important to understand ok. So far I have not discussed what polar codes are. In the next slide onwards we will start discussing what polar codes actually are.