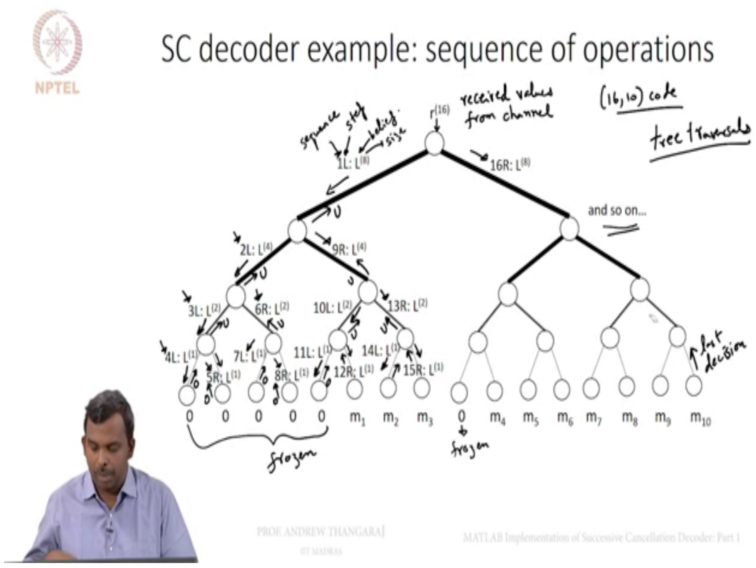
LDPC & Polar codes in 5G Standard
Prof. Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology Madras
MATLAB Implementation of Successive Cancellation Decoder Part 1

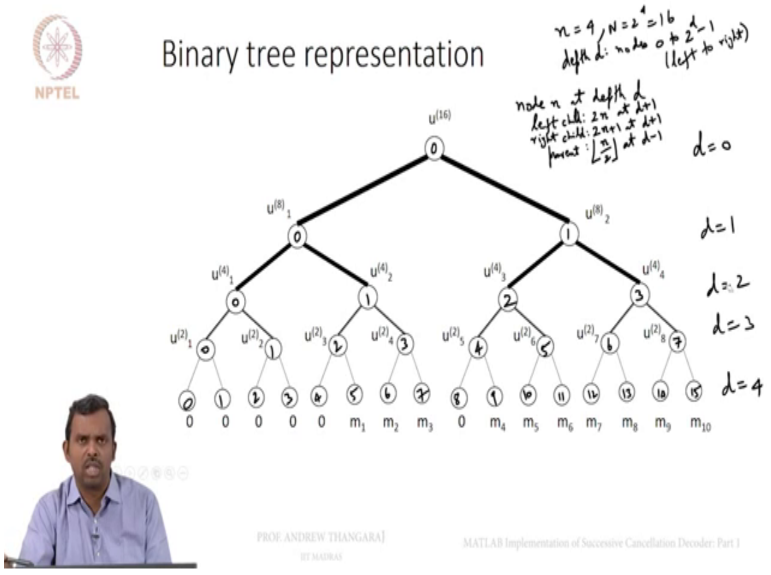
(Refer Slide Time: 0:20)



Okay hello and welcome to this lecture and in this lecture we will write a simple piece of MATLAB code for implementing this successive cancellation decoder, once again I want to emphasise you not writing very efficient code which could be probably directly implemented on hardware or something, we are just writing something that works okay, so that you can play around with it and use it for other sort of things okay.

Okay, so what will it take to implement this decoder? Of course you need a lot of representation right, inside the code how will you represent etc.

(Refer Slide Time: 0:50)



Binary tree representation

First think you need to represent is the tree itself, so how do you represent a binary tree? So like I said, I will do it the first thing I will use is this notion of depth, so this has depth equals 0, this level and then you have depth equals 1 and depth equals 2, depth equals 3, depth equals 4 okay, so remember this one is n equals 4 and N equals to power 4 which is 16 okay, so that is this example, if you have a more general one the depth will go further and further.

So for instance if it is 1024 the depth will go from D equals 0 all the way up to D equals 10 okay, so depth is one thing, so this tells you what depth you are and then what about the node okay, the node numbering will start with 0 and then go, 01,0,1,2,3, and then 0,1,2,3,4,5,6,7 and then 0,1,2,3,4,5,6,7,8,9,10,11,12 ,13,14,15 okay, so these are the numbers okay, so at any depth D you have nodes 0 to 2 power D minus 1 okay, so it is easy to see, so even for D equals 0 this is true right.

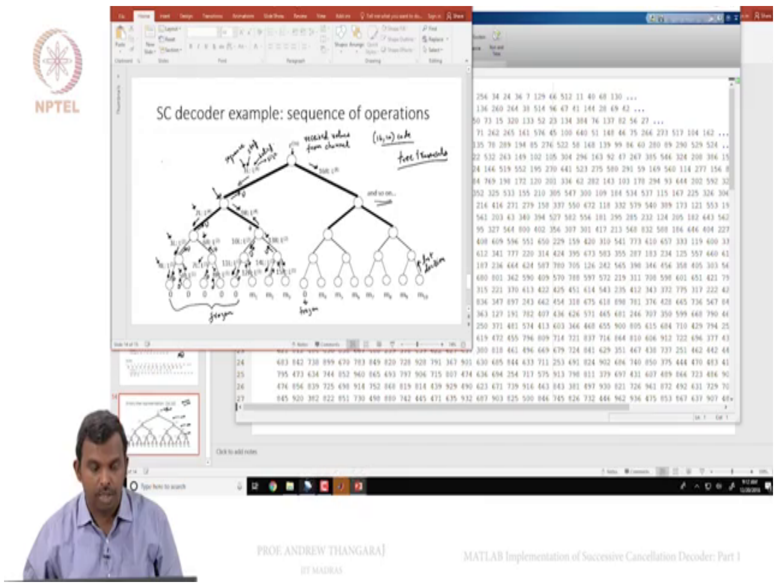
So if you have depth D you have node 0 to 2 power D minus 1 from left to right, that is one thing, so this is easy enough I have represented each node and then from each node I need to know what its parent is right, and I need to know what its left child is and what is right child is okay, so if you have node N at depth D okay, so what is the left child? What is the right child and what is the parent? That is my question, so you have, so avoid some trivial cases, so will assume it is some sort of an interior node, so that you do not have the issue of leaf or root, root has no parent, leaf has no children okay.

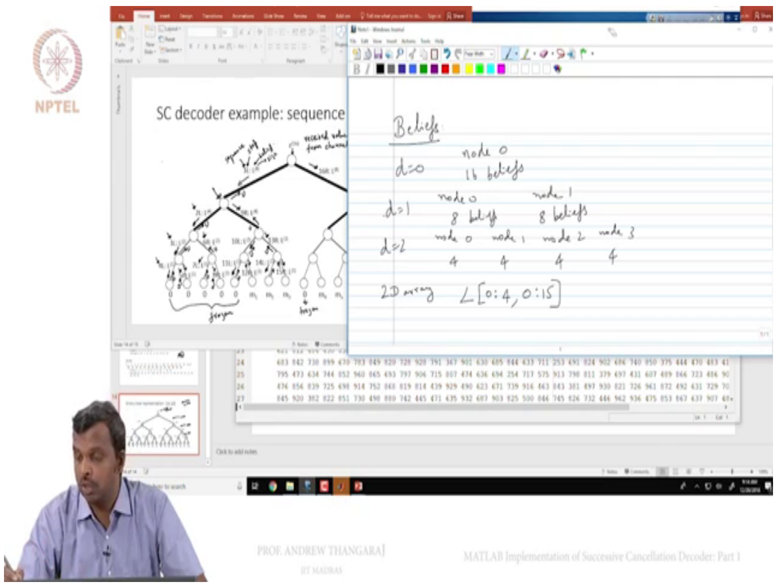
So will assume some interior node, so that node N and depth D, what is the left child? What is the right child and what is the parent? Okay, so you can look at it very very closely, there is a pattern here whenever you see the node; the left child is always multiplied by 2 okay, so this is going to go 2 times N okay, at D plus 1 right the next depth okay.

What about the right child? Again there is a very simple pattern, you can quickly observe it, it is possible to proof these things if you like, rigorous proofs 2N plus 1 at D plus 1 okay, so a left child is 2N, right child is 2N plus 1, so what is going to be the parent? Parent is floor of N by 2 at D minus 1 that it okay, so you see what floor is? Floor is you divide and throw where the reminder okay, so that is what happen if you want to go to the parent okay, so it is easy enough.

So once you have this notation of depth and node number from left to right, you can figure out easily what the left child is, right child is and the parent is okay, so this is something important we need to represent the binary tree and this is how we will represent the binary tree okay, so that is first part of representing the decoder, we are not done there is lots of other data that flows into the decoder, we need to represent that as well, we will see how to do that going along, so first thing we will use in the decoder is a representation for the nodes like this okay.

(Refer Slide Time: 4:35)


NPTEL
SC decoder example: sequence of operations
received values from channel
(16,10) code
tree traversal
frozen
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1



SC decoder example: sequence

Beliefs

d=0 node 0 16 beliefs

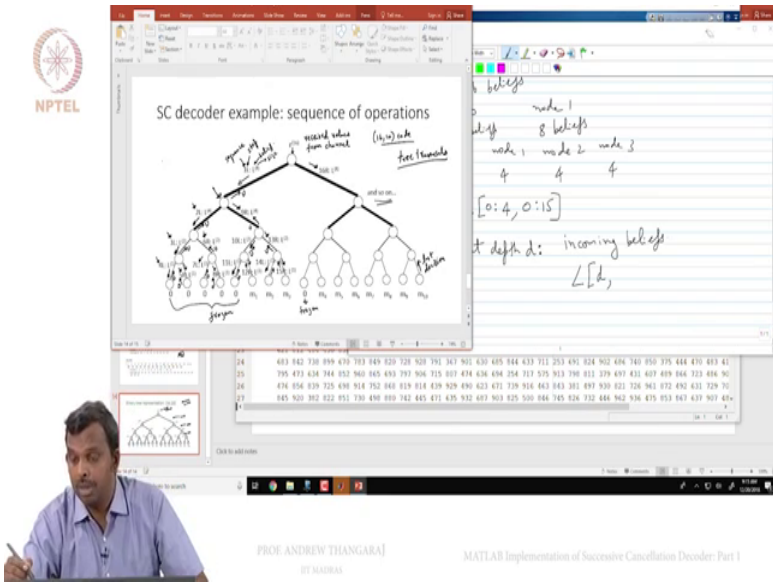
d=1 node 0 8 beliefs node 1 8 beliefs

d=2 node 0 4 node 1 4 node 2 4 node 3 4

2D array $L[0:4, 0:15]$

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



SC decoder example: sequence of operations

beliefs

node 1 8 beliefs

node 1 node 2 node 3

4 4 4

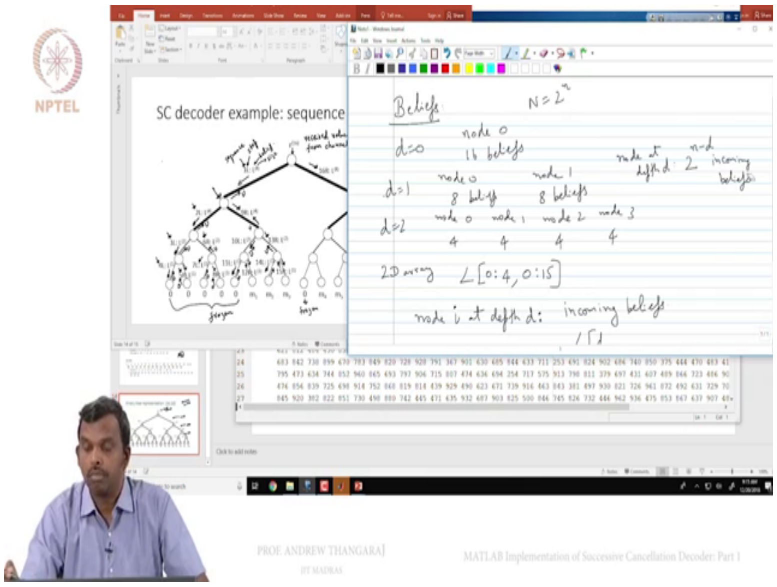
$[0:4, 0:15]$

depth d: incoming beliefs

$L[d,$

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



SC decoder example: sequence

Beliefs $N = 2^n$

d=0 node 0 16 beliefs

node at depth d: 2^{n-d} incoming beliefs

d=1 node 0 8 beliefs node 1 8 beliefs

d=2 node 0 4 node 1 4 node 2 4 node 3 4

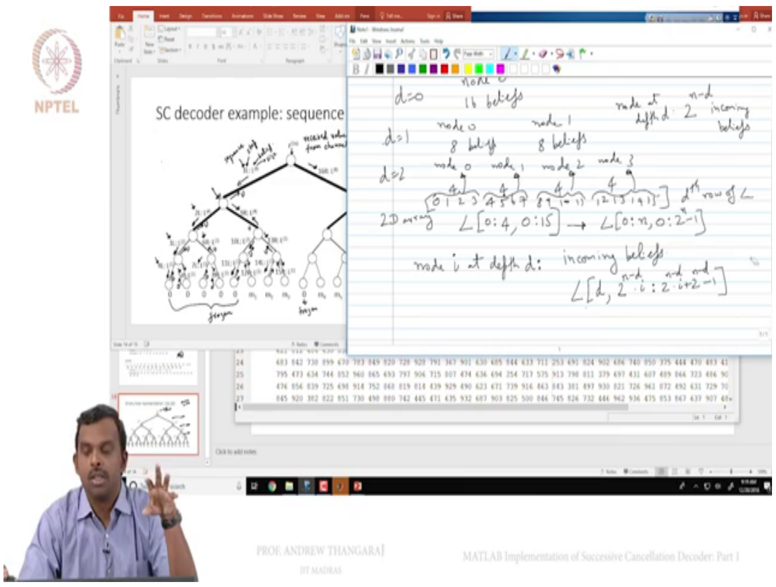
2D array $L[0:4, 0:15]$

node i at depth d: incoming beliefs

$L[d$

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1


SC decoder example: sequence

node 0

$d=0$ 16 beliefs

node at depth d: 2^{n-d} incoming beliefs

$d=1$ node 0 8 beliefs node 1 8 beliefs

$d=2$ node 0 node 1 node 2 node 3

$[\underbrace{0\ 1\ 2\ 3}_{4}\ \underbrace{4\ 5\ 6\ 7}_{4}\ \underbrace{8\ 9\ 10\ 11}_{4}\ \underbrace{12\ 13\ 14\ 15}_{4}]$ d^{th} row of L

2D array $L[0:4, 0:15] \rightarrow L[0:n, 0:2^n-1]$

node i at depth d: incoming beliefs

$L[d, 2^{n-d} \cdot i : 2^{n-d} \cdot i + 2^{n-d} - 1]$

PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

And the next thing is the messages okay, so what are the messages that need to be stored, so if you remember in the encoder, we had U at the bottom and then it got pushed into the next step, next step, next step and all that till it got to the root and there was no need to store any intermediate things, it was just going in one direction but the decoder does not go in one direction okay.

Excuse me, the decoder does not go in one direction, it does not go just down okay, it goes down then it comes back up and remember when it comes back up every node still needs to remember what is its incoming message was okay, you cannot forget that because of for instance here if you look at this node okay, I first receives 8 beliefs okay, these are some eight numbers okay, it processes it and sends out 4 beliefs to its left child, so it cannot forget its incoming 8 beliefs, why is that? Because when the left child send back its decision, it uses the incoming 8 beliefs and computes the G function okay.

So you have to do that okay but of course when both the left and right child have send their decisions you can forget about what is their okay, so it is possible, so you can, you do not have to store everything but we will be a bit relaxed about it and we will store all the messages that are flowing in every depth okay, so if you see here, you have depth 0 okay, you have beliefs, there are 16 beliefs okay, maybe I should write it a bit differently.

So what I will do is, I will keep this as a shorter window, not a full screen window can also open up journal here so that I can do something, I can write on journal as well and then you will see something little bit better okay, so let us do this okay, so this is a slide, so what about the received value, so you if you look at, this is the belief okay, so you have to store the beliefs, so how do you store the beliefs in the decoders, so if you look at depth 0 okay and node 0 have 16 beliefs incoming okay it needs to store all that.

Than if you look at depth 1, node 0 and depth 1 has 8 beliefs and then node 1 has 8 beliefs, these are all incoming, node 1 is a sort of node immediate, eventually it happens but it has 8 beliefs, likewise said D equals 2 you can see node 0, node 1, node 2, node 3, they have 4 beliefs incoming each okay, so basically in every depth you have 16 beliefs okay, in every depth you have 16 beliefs, they are associated with different nodes but the total is 16 okay.

So what we will do is, we will have two dimensional array L of let us say in this case 0: L I use this notation is 0:3 okay and then actually 4 right, so it go all the way up to 4 and then 0:15 okay, this will be the 2 minus D array but remember I am indexing from 0 and Matlab you have to do that minor adjustment to index from 1, so I will just, I will use 0 for now and you will see later on it easy to make that adjustment in Matlab okay.

So I have 2 minus D array which stores all this beliefs okay, so that is important, so once you store this beliefs, you can associate the nodes with them, so for instance node N at depth D okay, its incoming beliefs, these are all incoming beliefs okay are in L of D, okay four node N it will be, so 2 power D if you remember is the incoming, so maybe I should say that as well right maybe say that, for a node at depth D the incoming belief I mean, so I have done that here, so it is basically 2 power D right.

So node at depth D has, it is not 2 power D, it is 2 power N minus D okay, so let me just use this notations 2 power N okay, so I am using too many Ns here, so maybe I make this node I okay, so N is 2 power N okay, so if you have depth D node 0 has 16 beliefs right, so node at depth D has 2 power N minus D incoming beliefs okay, so hopefully that is clear to you have and so if you look at a node I at depth D the array entry it will correspond to its L of D, it will be in the Dth row and then what will be the columns that will be important.

So it will be 2 power N minus D okay, so times I:2 power N minus D times I plus 2 power N minus D minus 1 okay, so this will be the 2 power N minus D beliefs associated with the node I okay, so hopefully that is clear to you, so it for instance maybe I can use this example, so if you have L here of this, if you think offer array here so it will have the entry 0,1,2,3 associated with node 0 and then 4,5,6,7 these four associated with node 1 okay and then 8,9,10,11 this four are associated with node 2 and then 12,13,14,15 this four are associated with node 3 okay, so this is Dth row of L okay, hopefully that is easy enough to see

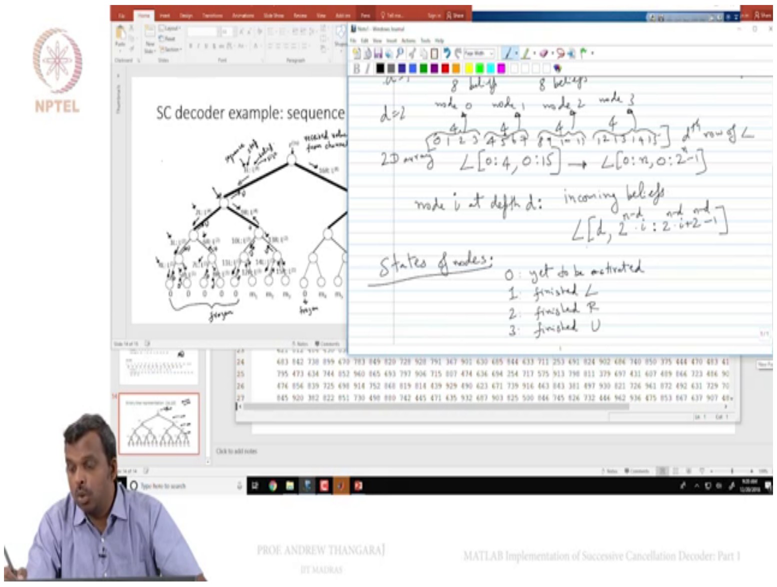
So you just group all of them together in one array but you associate different blocks to different nodes, so this is just for ease of storage so that we can refer to this when we want okay, hopefully this is clear, so we will use this array L which will be a two dimensional array, so in general it will be L of 0 colon N, 0 colon 2 power N minus 1 okay, so that will be the size of this array and the depth D is associated with 2 power N minus D incoming beliefs and how do you store those two party I mean you store it as one row vector but different chunks of it you associate with different nodes okay.

So this is how we do the beliefs okay, so we have represented the tree we know how to go from one node to the other in either left child, right child or parent, we have figure out how to store the beliefs okay, maybe not the most efficient, I am sure you can find a lot holes in how you can simplify the storage, you do not need to store is much etc, while like I said I am not trying to write the most efficient code here and the third piece of information that we need is some sort of a status for the node okay.

So if you remember a node comes into operation only once in a while, it gets activated and it can be in different states right, the state of a node it may be never mean, maybe it never been activated so far, maybe it has finished step L, it is waiting for step R maybe it has finished step R also it is waiting for step U maybe it has finish step U also, it just done okay, so there are multiple states of the nodes okay.

So for every node we have to store the state okay, so like I said there are three things you have to worry about, one is representing the binary tree being able to traverse we figure that out, next is storing the beliefs all through the tree and we are doing some extra storage, it is being very generous here and then the third part is storing the states of the node, every node will be indifferent state as it goes through the decoder okay, so you have to store the state okay.

(Refer Slide Time: 13:54)


SC decoder example: sequence

8 beliefs 8 beliefs

$d=2$ node 0 node 1 node 2 node 3

$[\underbrace{0\ 1\ 2\ 3}_{4}\ \underbrace{4\ 5\ 6\ 7}_{4}\ \underbrace{8\ 9\ 10\ 11}_{4}\ \underbrace{12\ 13\ 14\ 15}_{4}]$ d^{th} row of L

2D array $L[0:4, 0:15] \rightarrow L[0:n, 0:2^n-1]$

node i at depth d: incoming beliefs

$L[d, 2^{n-d}\cdot i : 2^{n-d}\cdot i + 2^{n-d} - 1]$

States of nodes:

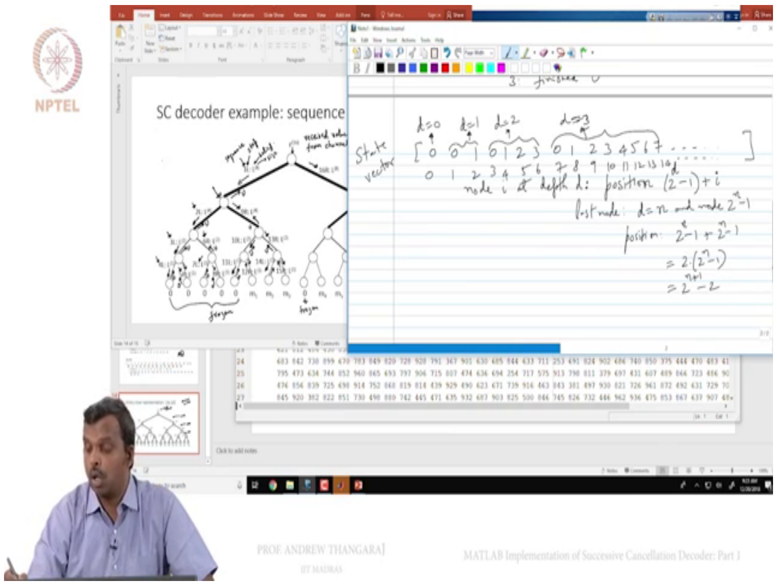
0: yet to be activated

1: finished L

2: finished R

3: finished U

PROF. ANDREW THANGARAJ IIT MADRAS MATLAB Implementation of Successive Cancellation Decoder: Part 1


SC decoder example: sequence

3: finished U

$d=0$ $d=1$ $d=2$ $d=3$

State vector $[\ 0\ \ \underbrace{0\ 1}\ \ \underbrace{0\ 1\ 2\ 3}\ \ \underbrace{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\ \ldots\ ]$

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

node i at depth d: position $(2^d-1)+i$

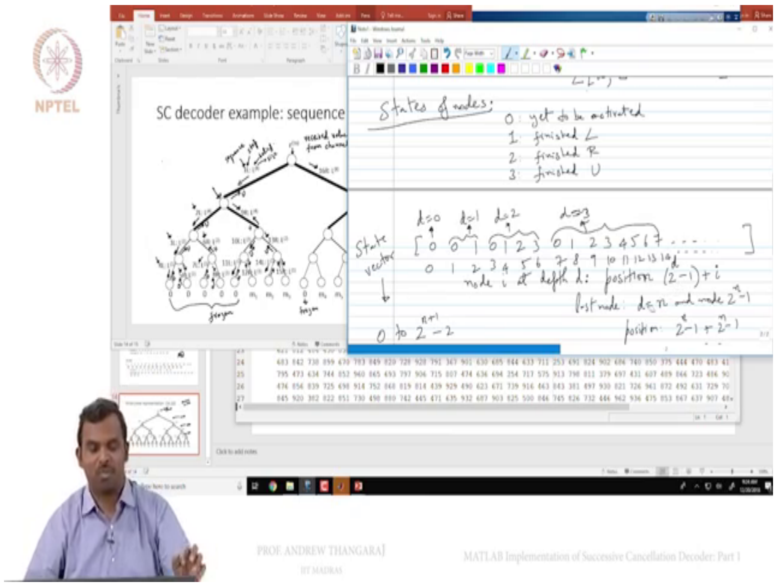
last node: $d=n$ and node 2^n-1

position: 2^n-1+2^n-1

$=2\cdot(2^n-1)$

$=2^{n+1}-2$

PROF. ANDREW THANGARAJ IIT MADRAS MATLAB Implementation of Successive Cancellation Decoder: Part 1


SC decoder example: sequence

States of nodes:

0: yet to be activated

1: finished L

2: finished R

3: finished U

$d=0$ $d=1$ $d=2$ $d=3$

State vector $[\ 0\ \ \underbrace{0\ 1}\ \ \underbrace{0\ 1\ 2\ 3}\ \ \underbrace{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}\ \ldots\ ]$

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

node i at depth d: position $(2^d-1)+i$

last node: $d=n$ and node 2^n-1

position: 2^n-1+2^n-1

0 to $2^{n+1}-2$

PROF. ANDREW THANGARAJ IIT MADRAS MATLAB Implementation of Successive Cancellation Decoder: Part 1

So states, we will use a slightly different notation just to be a bit more efficient than just you know just using some big 2 minus D array, we could use a big 2 minus D array as well but I will be a little bit more efficient the nodes to make things easy okay, so let us first summarise what the states are 0 is yet to be activated okay, 1 is finished L okay, 2 is finished, R3 is finished U okay, this is just my notation there are 4 states to the nodes, it could be 0 not yet activated, so initially all states will be 0 means all nodes will be in state 0 okay and then once you have done L it goes on to state 1 and once you have done R for that node it goes into state 2, once you have done U for that node, unit what is LR and UR, the three different steps that we do on the node goes into state 3 okay.

Once it goes into state 3 it is done okay, so this is how you store the node and then how I going to store the states okay, so how do I store the states, so maybe I do something slightly efficient here instead of being very very inefficient, so let us see so we have, remember so at depth 0 you have only 0 okay, so I will store the states in in one linear array okay, so initially I have depth 0 and then I have, this is depth 0 then I have depth 1, at depth 1 I have only 2 nodes right, 0 and 1, so this 2 are depth 1 and then I have depth 2, I have 0, 1, 2, 3 okay and then I have depth 3, I have 0, 1, 2, 3, 4, 5, 6, 7 and so on right.

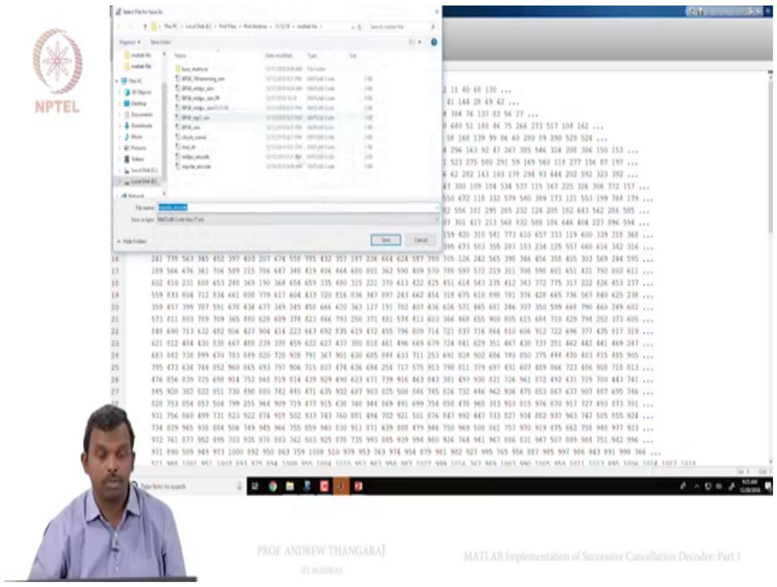
So this is what happens and so on, so in general node I at depth D will be at position, the position is, if you look at depth 3 it has 7 before, so it is going to be 2 power D minus 1 okay, so that is 7 plus I okay, so remember the overall count is 0 or 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 and so on okay that I miss anything okay, is one thing I miss anything, so it is goes on like this okay, so again I remember I am indexing from 0, this is one adjustment that you will have to make when we go to MATLAB change that.

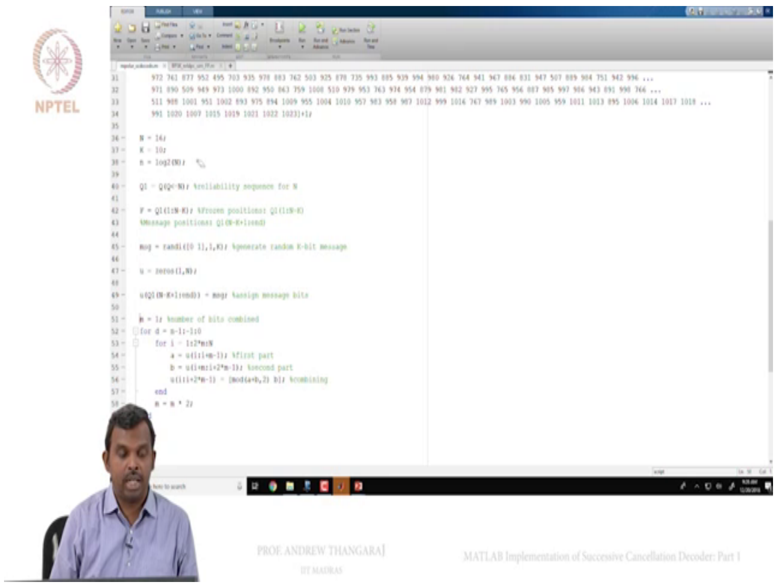
So node I at depth D is at position is 2 power D minus 1 plus I okay, so if you think of depth 2 and node 2 it will be 3plus2,5, so you will get the correct numbers, node 7 at depth 3 it will be 7plus7 which is 14 okay, so all of that will happen this is, so this is the state vector okay and what will be the total length you can compute, so what is the last node D equals N, the last node will be D equals N right depth of N and node 2 power N minus 1 right, so node 2 power N minus1 it depth 2 N, so that has position 2 power N minus 1 plus 2 power N minus1 which is 2 x 2 power N minus1 which is 2 power N plus 1 minus 2 okay, so the overall length, so it will go from 0 to 2 power N plus 1 minus 2 okay, so the length is this plus1 which is 2 power N plus 1 minus 1, so that is the total length we need.

So in fact there is the total number of nodes in the binary tree okay, so the leaf has 2 power minus, 2 power N then 2 power N by N minus1 so on till 1, so that will add up to 2 power N plus 1 minus 1 okay, so that is the total number of nodes, so will be a little efficient in storing the states, so once again what is the state vector, initially it will store all 0s and ones a particular node has completed a certain operation it will switch into either 1, 2 or 3 okay.

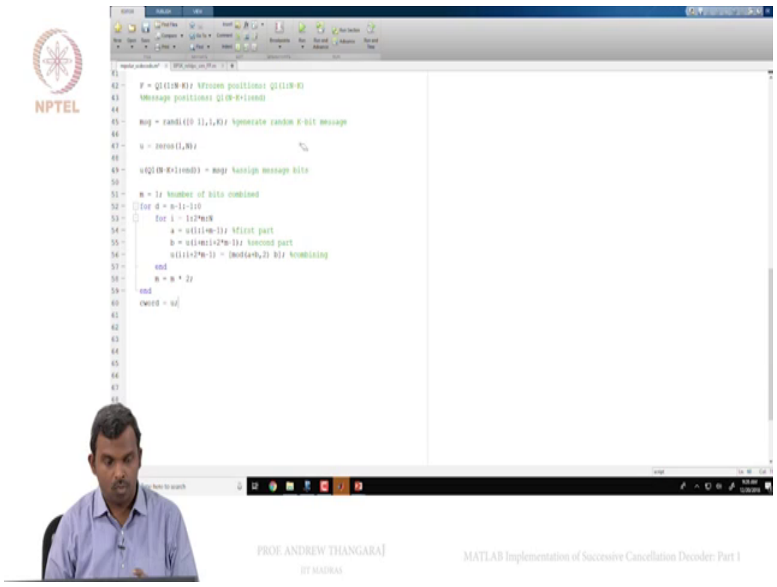
So with this we are ready to start writing the programs, so now it has been known how to store represent binary tree nodes and traverse from one to the other we know that, we know how to store the beliefs and how to access the beliefs corresponding to a particular node, we know how to store the states of the nodes and access the state of a particular node and we can start writing the decoder okay, let us see.

(Refer Slide Time: 19:20)


NPTEL
PROF ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part I


NPTEL
PROF ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part I

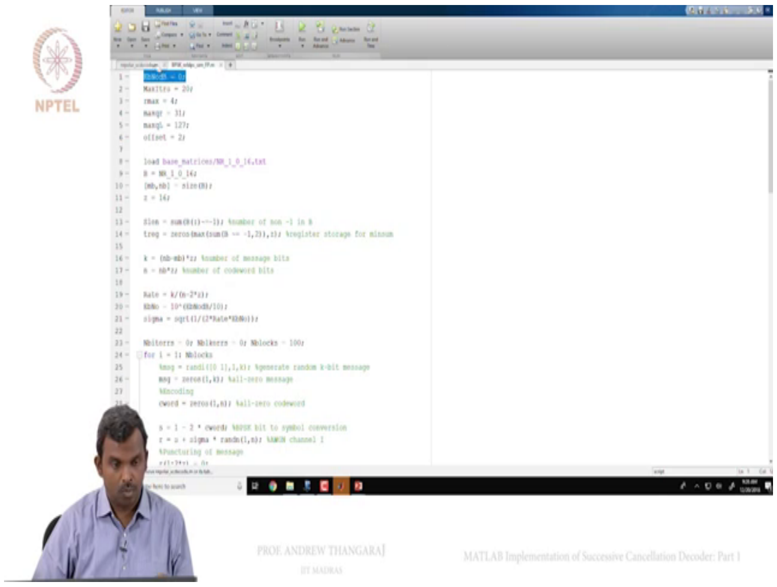
```
    972 761 877 952 495 703 935 978 883 762 503 925 878 735 993 885 939 994 980 926 764 941 967 886 831 947 507 889 984 751 942 996 ...
    971 890 509 949 973 1000 892 950 863 759 1008 510 979 953 763 974 954 879 981 982 927 995 765 956 887 985 997 986 943 891 998 766 ...
    511 988 1001 951 1002 893 975 894 1009 955 1004 1010 957 983 958 987 1012 999 1016 767 989 1003 990 1005 959 1011 1013 895 1006 1014 1017 1018 ...
    991 1020 1007 1015 1019 1021 1022 1023]+1;

N = 16;
K = 10;
n = log2(N);

Q1 = Q(Q<=N); %reliability sequence for N

F = Q1(1:N-K); %Frozen positions: Q1(1:N-K)
%Message positions: Q1(N-K+1:end)

msg = randi([0 1],1,K); %generate random K-bit message

u = zeros(1,N);

u(Q1(N-K+1:end)) = msg; %assign message bits

m = 1; %number of bits combined
for d = n-1:-1:0
    for i = 1:2*m:N
        a = u(i:i+m-1); %first part
        b = u(i+m:i+2*m-1); %second part
        u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
    end
    m = m * 2;
```



```matlab
F = Q1(1:N-K); %Frozen positions: Q1(1:N-K)
%Message positions: Q1(N-K+1:end)

msg = randi([0 1],1,K); %generate random K-bit message

u = zeros(1,N);

u(Q1(N-K+1:end)) = msg; %assign message bits

m = 1; %number of bits combined
for d = n-1:-1:0
    for i = 1:2*m:N
        a = u(i:i+m-1); %first part
        b = u(i+m:i+2*m-1); %second part
        u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
    end
    m = m * 2;
end
cword = u;
```

PROF. ANDREW THANGARAJ
IIT MADRAS

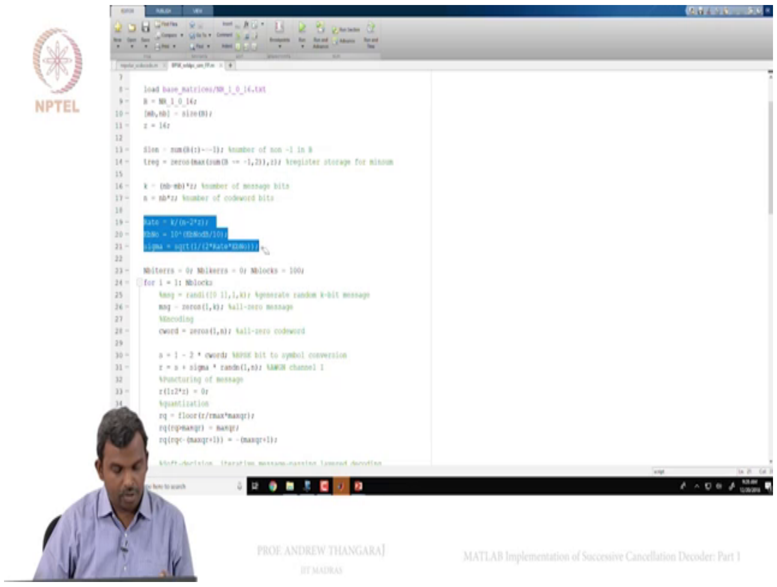
MATLAB Implementation of Successive Cancellation Decoder: Part 1



```matlab
EbNodB = 0;
MaxItrs = 20;
rmax = 4;
maxqr = 31;
maxqL = 127;
offset = 2;

load base_matrices/NR_1_0_16.txt
B = NR_1_0_16;
[mb,nb] = size(B);
z = 16;

Slen = sum(B(:)~=-1); %number of non -1 in B
treg = zeros(max(sum(B ~= -1,2)),z); %register storage for minsum

k = (nb-mb)*z; %number of message bits
n = nb*z; %number of codeword bits

Rate = k/(n-2*z);
EbNo = 10^(EbNodB/10);
sigma = sqrt(1/(2*Rate*EbNo));

Nbiterrs = 0; Nblkerrs = 0; Nblocks = 100;
for i = 1: Nblocks
    %msg = randi([0 1],1,k); %generate random k-bit message
    msg = zeros(1,k); %all-zero message
    %Encoding
    cword = zeros(1,n); %all-zero codeword

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,n); %AWGN channel I
    %Puncturing of message
    r(1:2*z) = 0;
```

PROF. ANDREW THANGARAJ
IIT MADRAS

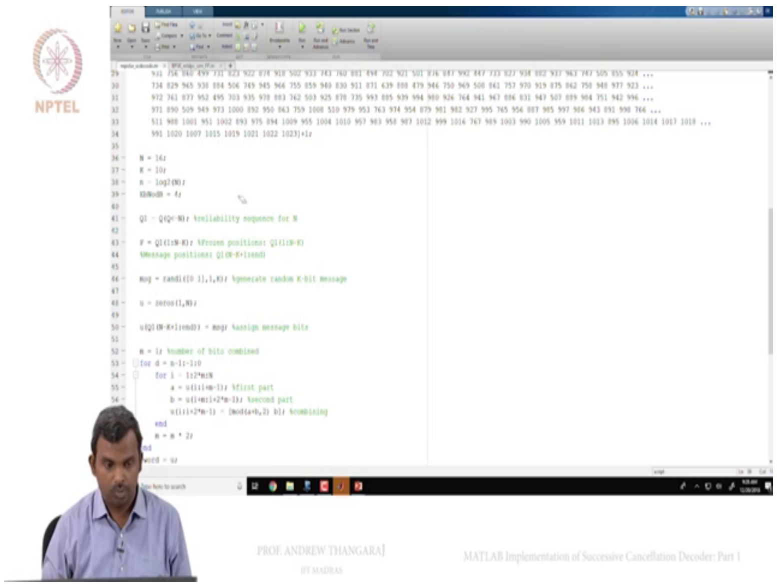
MATLAB Implementation of Successive Cancellation Decoder: Part 1



```matlab
load base_matrices/NR_1_0_16.txt
B = NR_1_0_16;
[mb,nb] = size(B);
z = 16;

Slen = sum(B(:)~=-1); %number of non -1 in B
treg = zeros(max(sum(B ~= -1,2)),z); %register storage for minsum

k = (nb-mb)*z; %number of message bits
n = nb*z; %number of codeword bits

Rate = k/(n-2*z);
EbNo = 10^(EbNodB/10);
sigma = sqrt(1/(2*Rate*EbNo));

Nbiterrs = 0; Nblkerrs = 0; Nblocks = 100;
for i = 1: Nblocks
    %msg = randi([0 1],1,k); %generate random k-bit message
    msg = zeros(1,k); %all-zero message
    %Encoding
    cword = zeros(1,n); %all-zero codeword

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,n); %AWGN channel I
    %Puncturing of message
    r(1:2*z) = 0;
    %quantization
    rq = floor(r/rmax*maxqr);
    rq(rq>maxqr) = maxqr;
    rq(rq<-(maxqr+1)) = -(maxqr+1);

    %Soft-decision, iterative message-passing layered decoding
```

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1


NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

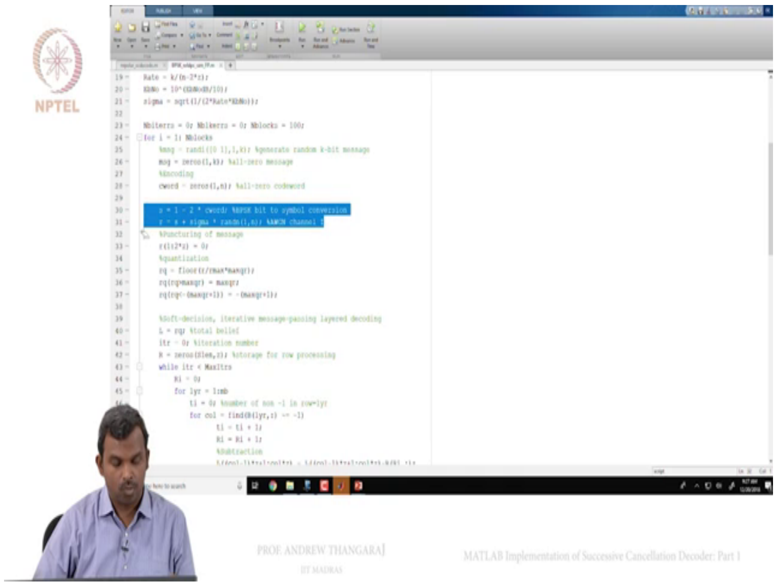
```
    991 1020 1007 1015 1019 1021 1022 1023]+1;

N = 16;
K = 10;
n = log2(N);
EbNodB = 4;

Q1 = Q(Q<=N); %reliability sequence for N

F = Q1(1:N-K); %frozen positions: Q1(1:N-K)
%Message positions: Q1(N-K+1:end)

msg = randi([0 1],1,K); %generate random K-bit message

u = zeros(1,N);

u(Q1(N-K+1:end)) = msg; %assign message bits

m = 1; %number of bits combined
for d = n-1:-1:0
    for i = 1:2*m:N
        a = u(i:i+m-1); %first part
        b = u(i+m:i+2*m-1); %second part
        u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
    end
    m = m * 2;
end
cword = u;
```

So what a MATLAB, so this is a the encoder that we had before, it had the reliability sequence, you remember the encoder, so what I am going to do is, I am going to save this as a decode, so I call it SE decode just to distinguish from other thing, so will have the reliability sequence, will keep it, we keep N as something small just so that we can do some debugging, maybe keep this as 10, we get the so frozen positions, so Q1 the frozen positions it is good to store the frozen positions explicitly.

So I will store them as F equals Q1 of 1 colon N minus K okay, so because in the decoder you to check if a particular index is frozen or not okay, so for that will keep this F as the list of frozen positions and after that the message and the U and the encoding can happen as before, so right now you finish with the encoding U is the code word and you can transmit okay.

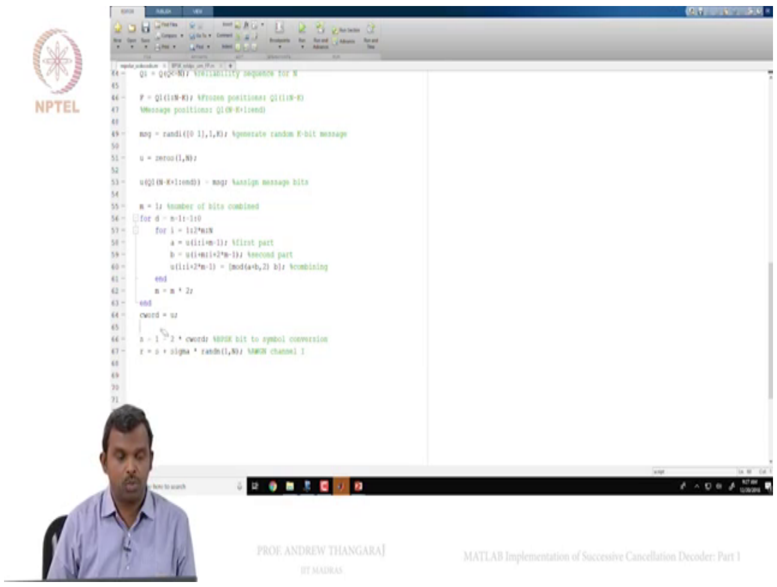
So let us transmit, for transmitting let just steal, so C words equals U, so for transmitting and all that setup let steal something from here, so will steal the EB over N not which has to be set up somewhere here, so 0 DB I keep it here, 4 DB, then you have this usual computations of rate EB over N not and this I have to change the rate calculation, what is it for us? Rate for us is K by N okay.

(Refer Slide Time: 21:19)



```
Rate = k/(n-2*z);
EbNo = 10^(EbNodB/10);
sigma = sqrt(1/(2*Rate*EbNo));

Nbiterrs = 0; Nblkerrs = 0; Nblocks = 100;
for i = 1: Nblocks
    %msg = randi([0 1],1,k); %generate random k-bit message
    msg = zeros(1,k); %all-zero message
    %Encoding
    cword = zeros(1,n); %all-zero codeword

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,n); %AWGN channel I
    %Puncturing of message
    r(1:2*z) = 0;
    %quantization
    rq = floor(r/rmax*maxqr);
    rq(rq>maxqr) = maxqr;
    rq(rq<-(maxqr+1)) = -(maxqr+1);

    %Soft-decision, iterative message-passing layered decoding
    L = rq; %total belief
    itr = 0; %iteration number
    R = zeros(Rlen,z); %storage for row processing
    while itr < MaxItrs
        Ri = 0;
        for lyr = 1:mb
            ti = 0; %number of non -1 in row=lyr
            for col = find(B(lyr,:) ~= -1)
                    ti = ti + 1;
                    Ri = Ri + 1;
                    %Subtraction
```

PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

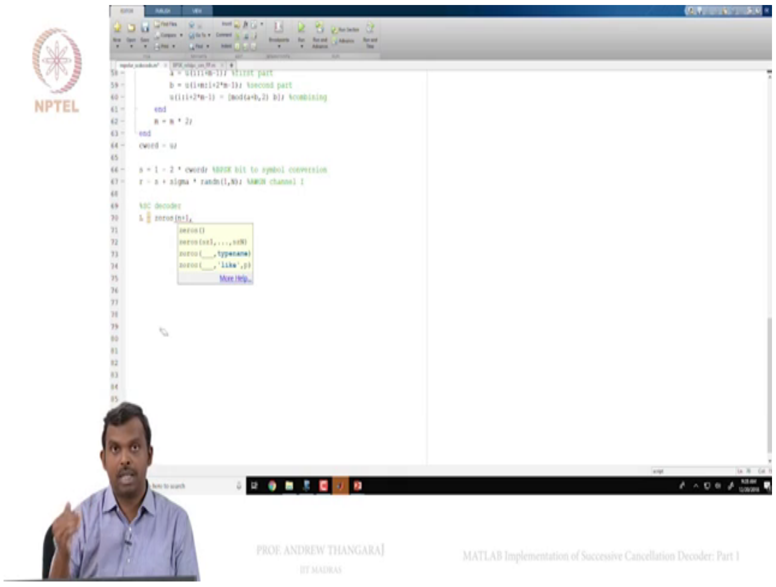


```
Q1 = Q(Q<=N); %reliability sequence for N

F = Q1(1:N-K); %Frozen positions: Q1(1:N-K)
%Message positions: Q1(N-K+1:end)

msg = randi([0 1],1,K); %generate random K-bit message

u = zeros(1,N);

u(Q1(N-K+1:end)) = msg; %assign message bits

m = 1; %number of bits combined
for d = n-1:-1:0
    for i = 1:2*m:N
        a = u(i:i+m-1); %first part
        b = u(i+m:i+2*m-1); %second part
        u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
    end
    m = m * 2;
end
cword = u;

s = 1 - 2 * cword; %BPSK bit to symbol conversion
r = s + sigma * randn(1,N); %AWGN channel I
```

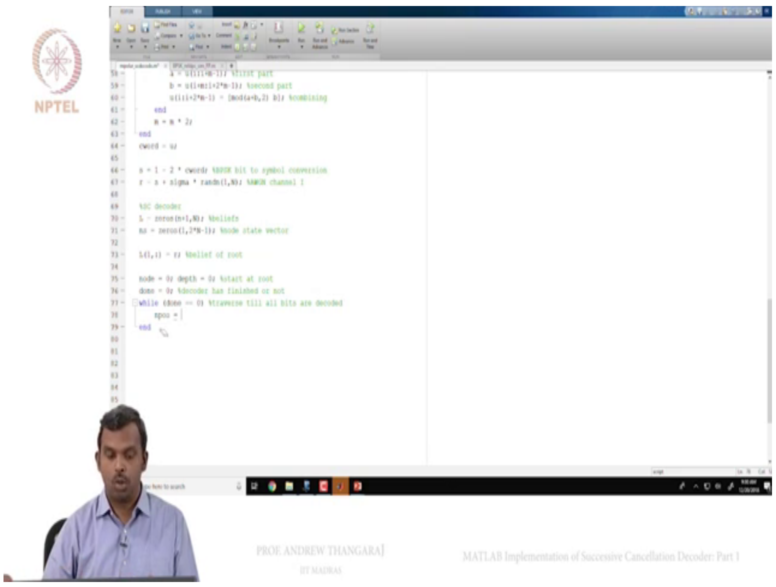
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

So I have done by sigma, once I have my sigma, I have my code word I can do transmission okay, so we going to transmit once you have the code word okay, so R is there, that is my received value okay but so I guess code word of length capital N that is one change it is needed okay that is it, so we are done, we have, we are ready to start the decoder.

(Refer Slide Time: 21:57)



```
        a = u(i:i+m-1); %first part
        b = u(i+m:i+2*m-1); %second part
        u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
    end
    m = m * 2;
end
cword = u;

s = 1 - 2 * cword; %BPSK bit to symbol conversion
r = s + sigma * randn(1,N); %AWGN channel I

%SC decoder
L = zeros(n+1,
```



```
        a = u(i:i+m-1); %first part
        b = u(i+m:i+2*m-1); %second part
        u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
    end
    m = m * 2;
end
cword = u;

s = 1 - 2 * cword; %BPSK bit to symbol conversion
r = s + sigma * randn(1,N); %AWGN channel I

%SC decoder
L = zeros(n+1,N); %beliefs
ns = zeros(1,2*N-1); %node state vector

L(1,:) = r; %belief of root

node = 0; depth = 0; %start at root
done = 0; %decoder has finished or not
while (done == 0) %traverse till all bits are decoded
    npos =
end
```

PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

So we have encoding, the reliability sequence big fact one from the standard, we setup the simulation, the frozen bits, code word and we are transmitting good, we are ready to go, so now the decoder, let us setup the storage, so L if you remember is initially put 0s here, so what is the total thing I needed, I need N plus 1 rows if you remember that the depth that we had, depth goes from 0 to N, so you need N plus 1 and on this site it just N always right and then the state vector okay I call it NS.

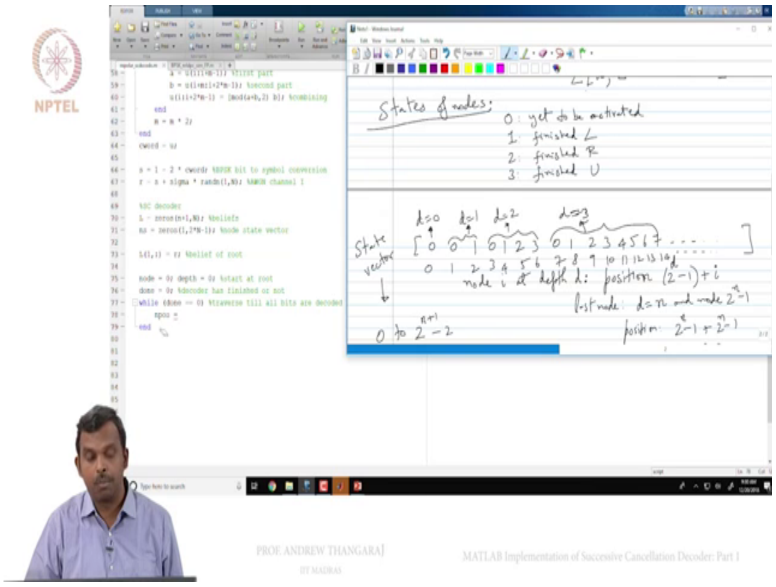
So these are the beliefs, so this is the state vector, node state, I am calling it NS, this will be 0s of its 2 par N plus 1 minus 1, so that is 1, 2 into N minus1 right, so that is the state vector, node state vector okay, so initially it will all be 0, once you have the R you can initialise L of

1, colon equals R okay, so that is the belief of root okay, so that is true and then the node, I will you simply node, node equals 0 and depth equals 0 okay.

So start at root, okay maybe this one put on top okay, that is it quick to start, I will also have a variable call done which will be 0 now, so just will just indicate that the decoder is finished or not, so initially is 0 which means it is not finish and my loop here will simply say while than equal to equal to 0 okay, so if it is 0 then you keep doing the loops, so now this loop is just the traversal loop okay just to keep the traversal going, so remember I am starting at the root and I have keep traversing.

Once I have decoded all the messages I can stop but till then I have to keep traversing okay, so what do we do first? First I know what node I am at? I am at node and depth equals 0, node equals 0, I have to first check the state okay, depending on the state have to do something, so if, let me calculate the node position.

(Refer Slide Time: 24:53)



```
            a = u(i:i+m-1); %first part
            b = u(i+m:i+2*m-1); %second part
            u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
        end
        m = m * 2;
    end
    cword = u;

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,N); %AWGN channel I

    %SC decoder
    L = zeros(n+1,N); %beliefs
    ns = zeros(1,2*N-1); %node state vector

    L(1,:) = r; %belief of root

    node = 0; depth = 0; %start at root
    done = 0; %decoder has finished or not
    while (done == 0) %traverse till all bits are decoded
        npos =
    end
```

States of nodes:

0: yet to be activated
1: finished L
2: finished R
3: finished U

State vector

$$[\underbrace{0}_{d=0}\ \underbrace{0\ 1}_{d=1}\ \underbrace{0\ 1\ 2\ 3}_{d=2}\ \underbrace{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}_{d=3} \ldots\ ]$$

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

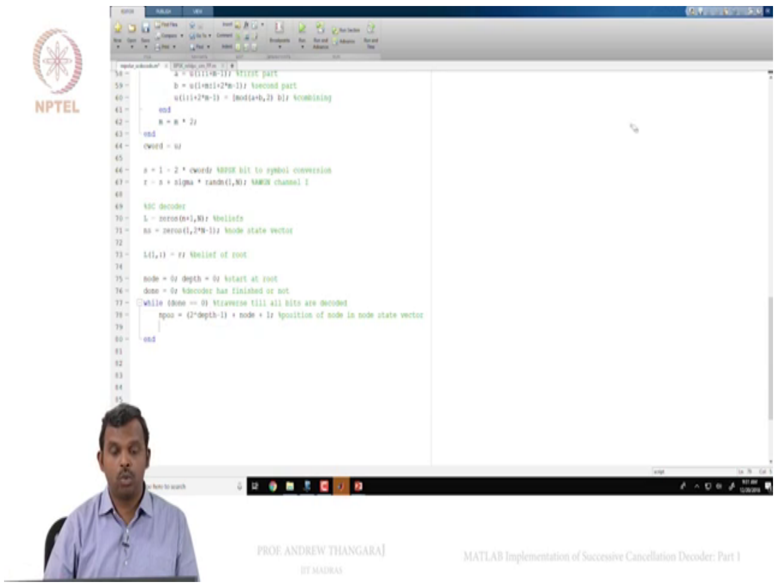
node i at depth d: position $(2^d-1)+i$

last node: $d=n$ and node 2^n-1

position: 2^n-1+2^n-1

0 to $2^{n+1}-2$

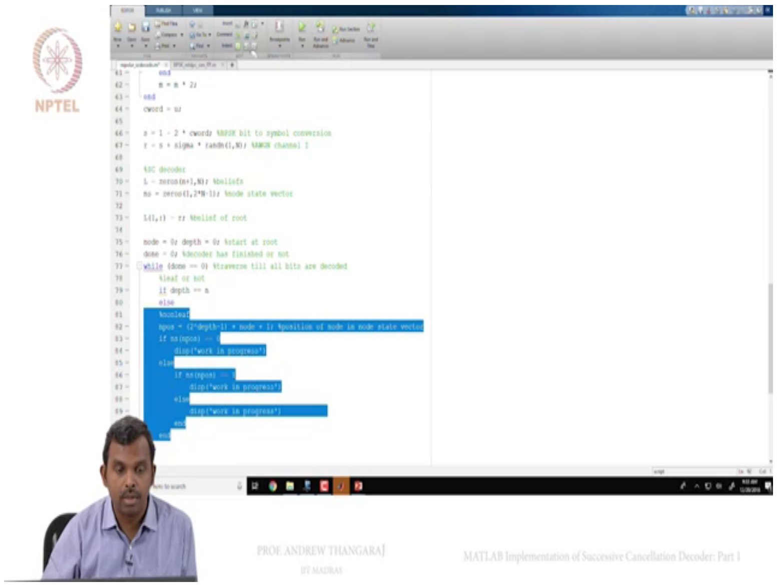
NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part I



```matlab
            a = u(i:i+m-1); %first part
            b = u(i+m:i+2*m-1); %second part
            u(i:i+2*m-1) = [mod(a+b,2) b]; %combining
        end
        m = m * 2;
    end
    cword = u;

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,N); %AWGN channel I

    %SC decoder
    L = zeros(n+1,N); %beliefs
    ns = zeros(1,2*N-1); %node state vector

    L(1,:) = r; %belief of root

    node = 0; depth = 0; %start at root
    done = 0; %decoder has finished or not
    while (done == 0) %traverse till all bits are decoded
        npos = (2^depth-1) + node + 1; %position of node in node state vector

    end
```

PROF. ANDREW THANGARAJ
IIT MADRAS

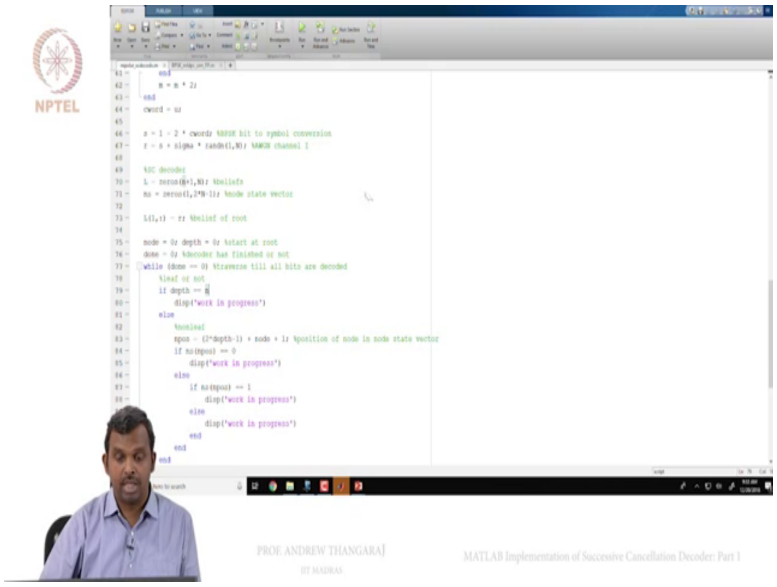
MATLAB Implementation of Successive Cancellation Decoder: Part 1



```matlab
        end
        m = m * 2;
    end
    cword = u;

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,N); %AWGN channel I

    %SC decoder
    L = zeros(n+1,N); %beliefs
    ns = zeros(1,2*N-1); %node state vector

    L(1,:) = r; %belief of root

    node = 0; depth = 0; %start at root
    done = 0; %decoder has finished or not
    while (done == 0) %traverse till all bits are decoded
        %leaf or not
        if depth == n
        else
        %nonleaf
        npos = (2^depth-1) + node + 1; %position of node in node state vector
        if ns(npos) == 0
            disp('work in progress')
        else
            if ns(npos) == 1
                disp('work in progress')
            else
                disp('work in progress')
            end
        end
```

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1



```matlab
        end
        m = m * 2;
    end
    cword = u;

    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,N); %AWGN channel I

    %SC decoder
    L = zeros(n+1,N); %beliefs
    ns = zeros(1,2*N-1); %node state vector

    L(1,:) = r; %belief of root

    node = 0; depth = 0; %start at root
    done = 0; %decoder has finished or not
    while (done == 0) %traverse till all bits are decoded
        %leaf or not
        if depth == n
            disp('work in progress')
        else
            %nonleaf
            npos = (2^depth-1) + node + 1; %position of node in node state vector
            if ns(npos) == 0
                disp('work in progress')
            else
                if ns(npos) == 1
                    disp('work in progress')
                else
                    disp('work in progress')
                end
            end
        end
```

PROF. ANDREW THANGARAJ
IIT MADRAS

MATLAB Implementation of Successive Cancellation Decoder: Part 1

Node position if you remember I just wrote it down here 2 par D minus 1plus I okay, it is 2 power depth minus1plus node, and N plus1, why do I do plus1 that is the MATLAB adjustment for the position of node in node state better right, so that is the N Poss and then you have to do multiple things depending on whether or not N Poss is a, what is the state okay, so you can do a lot of if conditions here, it just few states only, if NS of N Poss equal to equal to 0 okay, so that is one condition else so for now I simply say display work progress will write it, else if, so it is good to end this else is, if NS of N Poss, why is it?

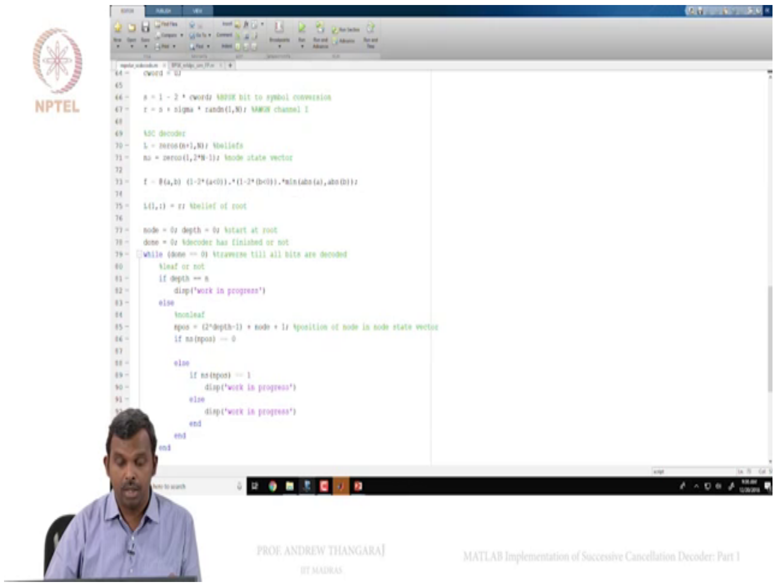
Okay, so I think I made a mistake here, there should be round bracket, NS of N Poss equal to equal to 1, you have to do something else if NS of N Poss equal to, actually else you do not need, means if it is not 0 or 1, it is going to be 2 and you know what to do, for now I simply say display work in progress, whenever it finishes, will finish okay alright.

So depending on where we are, we do different things that is good okay, then okay, so that is nice alright, so even before all of this I think one thing you can do is, you can check whether your leaf or not right, if you are the leaf than you know what to do, so this is assuming non-leaf okay.

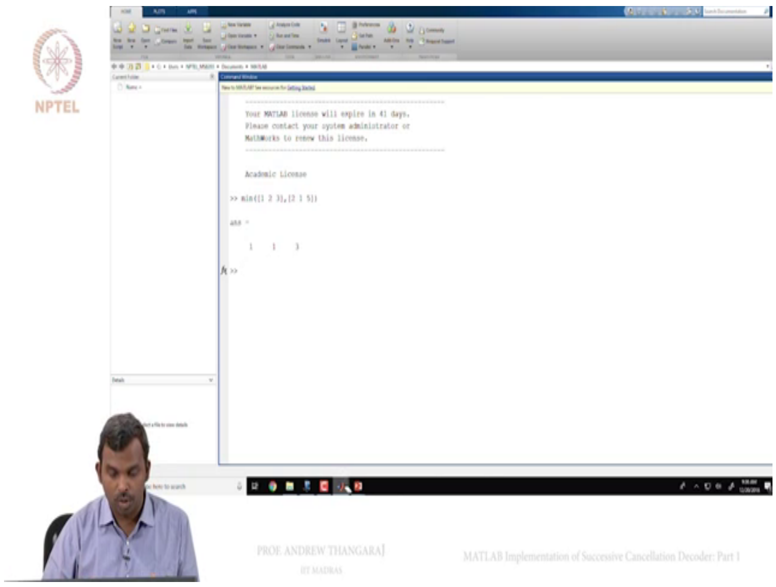
So if your leaf, there is a very simple thing to do here, so we will put that code in a little bit later so for now I am assuming, I am non-leaf okay, I am proceeding to do that part, if you leafs, so you have to check that, so how to check that? If depth equal to equal to N okay, so if you have reached the last one then you do something else right, so only if you are not so you to do this non-leaf stuff right.

So let us push this to the right and then get N here so that things work out, so I will just keep work in progress here so that we come back and read something there okay, so hopefully this is clear, so how does that operation works, so in every node I first check if I am leaf or not, if I am not leaf I have to do something, if I am leaf I just have to do you know, I have to make a decision and then do something, only if I am not leaf I do anything okay.

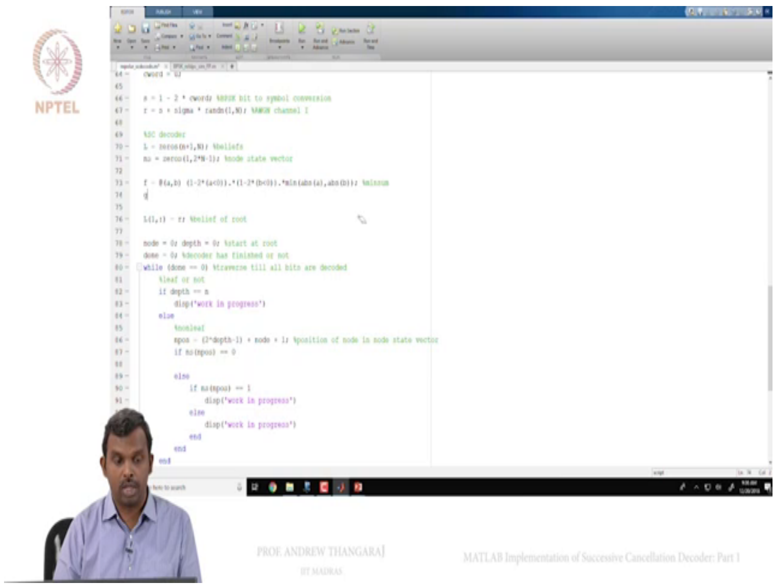
(Refer Slide Time: 28:37)


PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

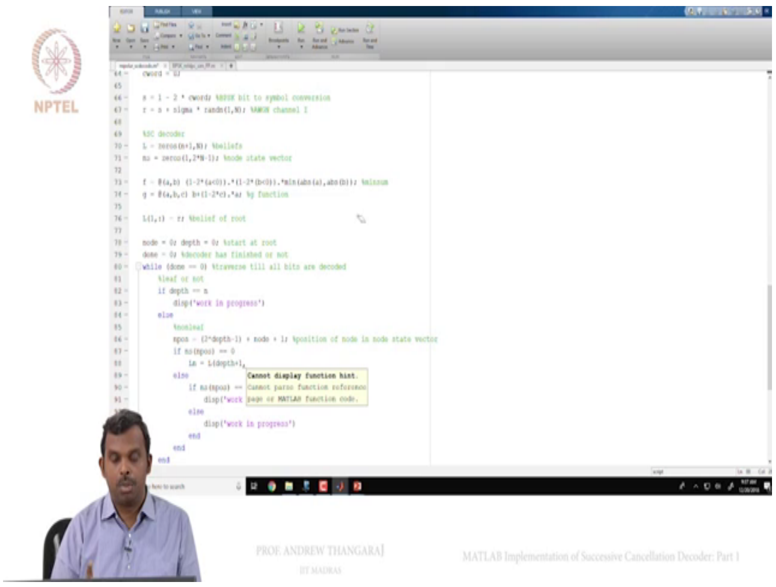
```
cword = u;

s = 1 - 2 * cword; %BPSK bit to symbol conversion
r = s + sigma * randn(1,N); %AWGN channel I

%SC decoder
L = zeros(n+1,N); %beliefs
ns = zeros(1,2*N-1); %node state vector

f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b));

L(1,:) = r; %belief of root

node = 0; depth = 0; %start at root
done = 0; %decoder has finished or not
while (done == 0) %traverse till all bits are decoded
    %leaf or not
    if depth == n
        disp('work in progress')
    else
        %nonleaf
        npos = (2^depth-1) + node + 1; %position of node in node state vector
        if ns(npos) == 0

        else
            if ns(npos) == 1
                disp('work in progress')
            else
                disp('work in progress')
            end
        end
    end
```


PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

```
    -----------------------------------------------
    Your MATLAB license will expire in 41 days.
    Please contact your system administrator or
    MathWorks to renew this license.
    -----------------------------------------------

    Academic License

>> min([1 2 3],[2 1 5])

ans =

     1     1     3

>>
```

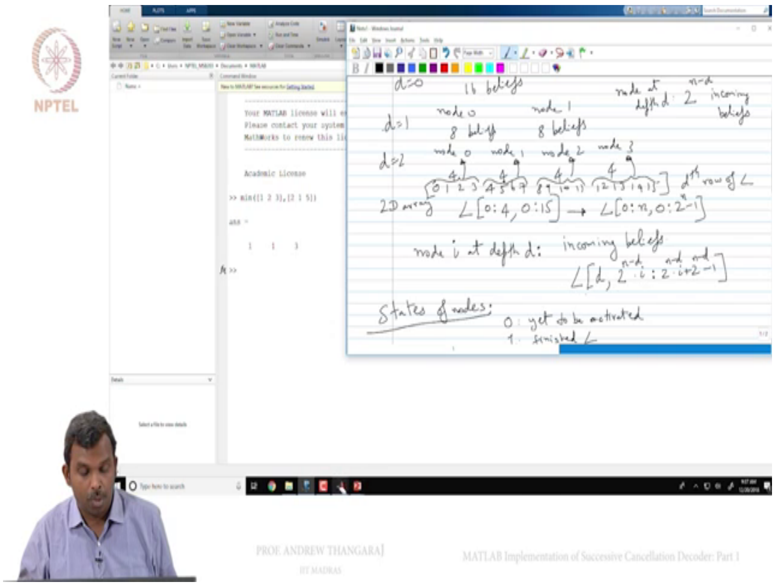

PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

```
cword = u;

s = 1 - 2 * cword; %BPSK bit to symbol conversion
r = s + sigma * randn(1,N); %AWGN channel I

%SC decoder
L = zeros(n+1,N); %beliefs
ns = zeros(1,2*N-1); %node state vector

f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
g

L(1,:) = r; %belief of root

node = 0; depth = 0; %start at root
done = 0; %decoder has finished or not
while (done == 0) %traverse till all bits are decoded
    %leaf or not
    if depth == n
        disp('work in progress')
    else
        %nonleaf
        npos = (2^depth-1) + node + 1; %position of node in node state vector
        if ns(npos) == 0

        else
            if ns(npos) == 1
                disp('work in progress')
            else
                disp('work in progress')
            end
        end
    end
```


PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

```
cword = []

s = 1 - 2 * cword; %BPSK bit to symbol conversion
r = s + sigma * randn(1,N); %AWGN channel I

%SC decoder
L = zeros(n+1,N); %beliefs
ns = zeros(1,2*N-1); %node state vector

f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
g = @(a,b,c) b+(1-2*c).*a; %g function

L(1,:) = r; %belief of root

node = 0; depth = 0; %start at root
done = 0; %decoder has finished or not
while (done == 0) %traverse till all bits are decoded
    %leaf or not
    if depth == n
        disp('work in progress')
    else
        %nonleaf
        npos = (2^depth-1) + node + 1; %position of node in node state vector
        if ns(npos) == 0
            Ln = L(depth+1,
        else
            if ns(npos) ==
                disp('work
            else
                disp('work in progress')
            end
        end
    end
```

```
Cannot display function hint.
Cannot parse function reference
page or MATLAB function code.
```


PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

```
Your MATLAB license will e
Please contact your system
MathWorks to renew this li

Academic License

>> min([1 2 3],[2 1 5])

ans =

     1     1     3

>>
```

d=0 16 beliefs

node at depth d: 2^{n-d} incoming beliefs

d=1 node 0 8 beliefs node 1 8 beliefs

d=2 node 0 node 1 node 2 node 3

$[\underbrace{0\ 1\ 2\ 3}_{4}\ \underbrace{4\ 5\ 6\ 7}_{4}\ \underbrace{8\ 9\ 10\ 11}_{4}\ \underbrace{12\ 13\ 14\ 15}_{4}]$ d^{th} row of L

2D array $L[0:4, 0:15] \rightarrow L[0:n, 0:2^{n}-1]$

node i at depth d: incoming beliefs $L[d, 2^{n-d}\cdot i : 2^{n-d}\cdot i + 2^{n-d} - 1]$

States of nodes:
0 : yet to be activated
1 : finished L

So let us start on this, so if I am not leaf I am in state 0, let us see how to write this up okay, so what do I do? I have to do minsum and for minsum we have this simple small little F function and the G function right, so one of the things I like to do is to write this F and G functions as this MATLAB in-line functions.

So we can write F as AB, so I do not want to use the MATLAB sign function, the problem with MATLAB sign function is it gives the 0 for 0, so that is not very nice so maybe we can do this modified thing right, so we can say 1 minus 2 times A less than 0 right, so if there is greater than or equal to 0, now maybe I should put here A less than 0, yes this is good because N if A is negative this is going to be minus 1, if a 0 or positive this is going to be plus 1 okay.

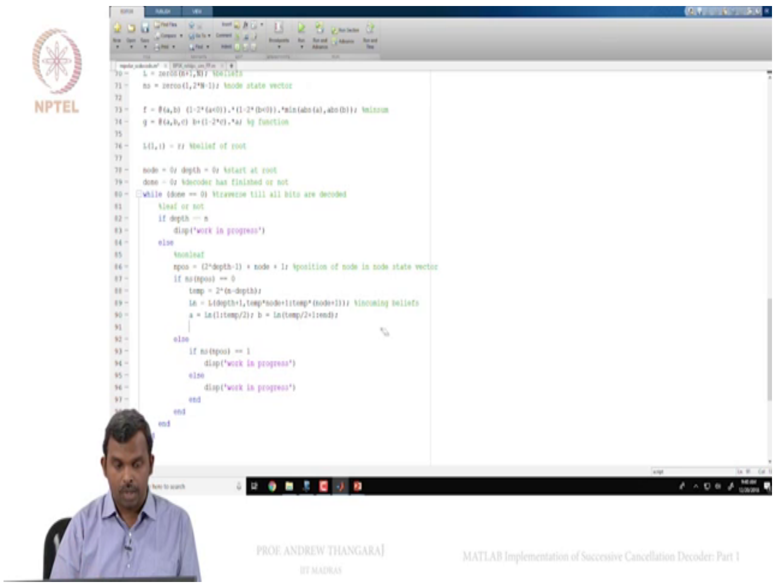
So this times 1 minus 2 times, so it is goes to assume that A and B will be vectors, so I will put a dot product here and then dot into min of ABS of A, ABS of B, hopefully this will work, this cut okay, so this looks okay enough for me sometimes I like to check if this will work min of for instance 1, 2, 3 2, 1, 5 hopefully will works okay, it is good to check once in a while that what you are writing is correct.

So this is F okay, so this is F function if you remember, it does the minsum okay, general vector also it will work, then I will also need the G, G has three inputs ABC and its output is considerably simpler, it is B plus 1 minus 2 into C, dot into A, this is the G function okay, so you remember the G function it does, R2 minus R1, R2 plus R1 depending on whether or not C is 0 or 1 okay, so that is G and F okay.

So once you write G and F, this is easy enough to write down, so we just have to figure out what A is and what B is? Right, so what is A? Remember for node and depth, we know what is the corresponding L value, so L node equals L of depth plus 1, Right you remember the little formula that I wrote down.

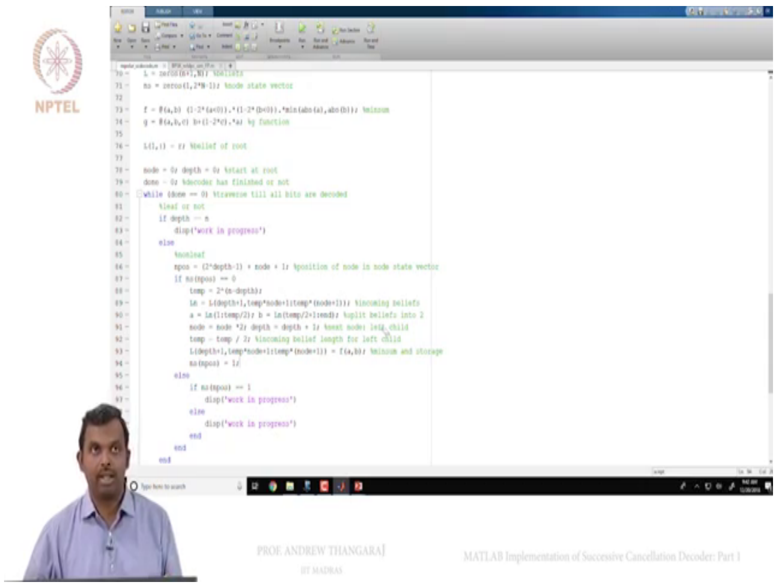
So its 2 par N minus D times I from then 2 par N minus D times I plus this guy right, so this 2 par N minus D keeps appearing quite prominently, so it is probably worth computing that first okay, so I call it N equals 2 power N minus depth okay and then this is going to be temp times node plus 1 colon temp times node plus temp so it is just little crop the node plus 1 okay.

(Refer Slide Time: 32:47)



```
L = zeros(n+1,N); %beliefs
ns = zeros(1,2*N-1); %node state vector

f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
g = @(a,b,c) b+(1-2*c).*a; %g function

L(1,:) = r; %belief of root

node = 0; depth = 0; %start at root
done = 0; %decoder has finished or not
while (done == 0) %traverse till all bits are decoded
    %leaf or not
    if depth == n
        disp('work in progress')
    else
        %nonleaf
        npos = (2^depth-1) + node + 1; %position of node in node state vector
        if ns(npos) == 0
            temp = 2^(n-depth);
            Ln = L(depth+1,temp*node+1:temp*(node+1)); %incoming beliefs
            a = Ln(1:temp/2); b = Ln(temp/2+1:end);

        else
            if ns(npos) == 1
                disp('work in progress')
            else
                disp('work in progress')
            end
        end
    end
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1



```
L = zeros(n+1,N); %beliefs
ns = zeros(1,2*N-1); %node state vector

f = @(a,b) (1-2*(a<0)).*(1-2*(b<0)).*min(abs(a),abs(b)); %minsum
g = @(a,b,c) b+(1-2*c).*a; %g function

L(1,:) = r; %belief of root

node = 0; depth = 0; %start at root
done = 0; %decoder has finished or not
while (done == 0) %traverse till all bits are decoded
    %leaf or not
    if depth == n
        disp('work in progress')
    else
        %nonleaf
        npos = (2^depth-1) + node + 1; %position of node in node state vector
        if ns(npos) == 0
            temp = 2^(n-depth);
            Ln = L(depth+1,temp*node+1:temp*(node+1)); %incoming beliefs
            a = Ln(1:temp/2); b = Ln(temp/2+1:end); %split beliefs into 2
            node = node *2; depth = depth + 1; %next node: left child
            temp = temp / 2; %incoming belief length for left child
            L(depth+1,temp*node+1:temp*(node+1)) = f(a,b); %minsum and storage
            ns(npos) = 1;
        else
            if ns(npos) == 1
                disp('work in progress')
            else
                disp('work in progress')
            end
        end
    end
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
MATLAB Implementation of Successive Cancellation Decoder: Part 1

So you can do a few charity checks for instance if depth is 0 node will also be 0 okay, so this temp will be just 1 okay, so at depth 0 I get 1, so node is 0 and temp is 1, so it will go from 1 to 1 okay, so thinks that okay because the first time, no I am sorry if depth is 0 temp is 2 power N okay, so it will go from 1 to 2 power N okay.

So I think that is okay temp is 2 power N, node is 0, so it will go from 1 to 2 parent, so it seems fine, so LN is my incoming beliefs okay, so I have pulled it out of the storage, the L is the storage throughout I pulled it out and got LN okay and then I can pull out these A and B or maybe I can write it out explicitly for the next one right, so maybe I should do the A and B, A is LN of, so what is the length of this? The length of this is temp okay.

So LN of 1 colon temp by 2, B is LN of okay, so I have pulled out the corresponding thing, so temp by 2 plus 1 colon end okay, so this is the A and B okay and so we are ready to write down now what the next node should be, node equals, so once, so if it is non-leaf and I am at state 0 I have to proceed to the left child, how do I go to the left child? Node is node into 2, depth is depth plus 1 right, so that is what we do and once you have the node, any node, you have a new temp okay, so you have gone down in-depth, so temp will become temp by 2 right.

So it is 1, one step down okay, right so initially I was at a certain the incoming belief length, the belief length is going down by 2 and then now I can assign what it should go in L? L of depth plus 1, the same thing temp into node plus 1 colon temp into node plus 1 equals F of A, okay that is it, so this was the, this is the next node split beliefs into 2, next node incoming belief length, next node remember is left child for left child than minsum, minsum and storage right.

So you have to do the minsum calculation and store it back also that is very important, so this is the step for state 0, of course you have to change the state NS of N Poss should become okay as soon I do this NS of N Poss should become 1 okay, so this is done okay, so hopefully you saw how this was done, so I am going to cut this lecture at this point, we will come back and in the next lecture see the other steps and how they work and also we will check how this is working correctly a little bit later but this is a good point to stop this lecture and do the coding in the remaining next lecture.