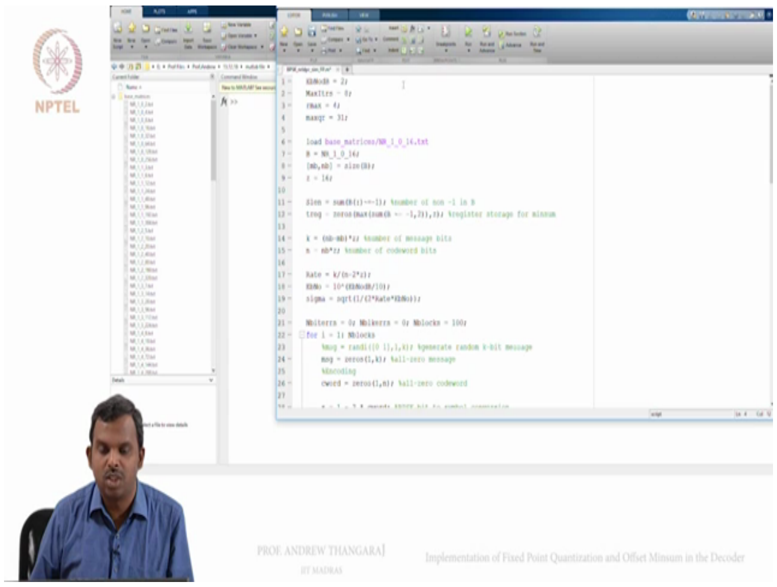
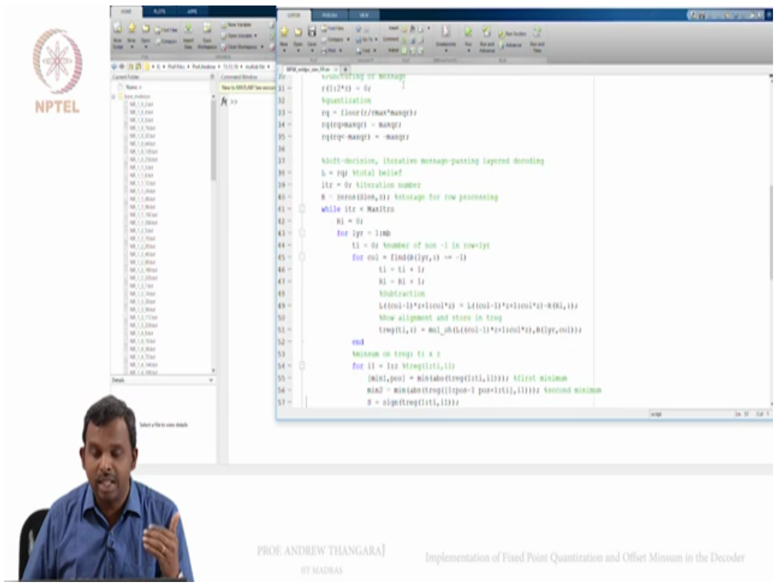
LDPC and Polar codes in 5G Standard
Professor Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology, Madras
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

(Refer Slide Time: 0:16)



```
EbNodB = 2;
MaxItrs = 8;
rmax = 4;
maxqr = 31;

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
```



```
    r(1:2*z) = 0;
    %quantization
    rq = floor(r/rmax*maxqr);
    rq(rq>maxqr) = maxqr;
    rq(rq<-maxqr) = -maxqr;

    %Soft-decision, iterative message-passing layered decoding
    L = rq; %total belief
    itr = 0; %iteration number
    R = zeros(Slen,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
                    L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)-R(Ri,:);
                    %Row alignment and store in treg
                    treg(ti,:) = mul_sh(L((col-1)*z+1:col*z),B(lyr,col));
            end
            %minsum on treg: ti x z
            for i1 = 1:z %treg(1:ti,i1)
                [min1,pos] = min(abs(treg(1:ti,i1))); %first minimum
                min2 = min(abs(treg([1:pos-1 pos+1:ti],i1))); %second minimum
                S = sign(treg(1:ti,i1));
```

Hello welcome to this lecture on the final lecture hopefully on LDPC decoding where we will convert the code to fixed point real quick and then do the offset minsum then with that we will close. So to convert the fixed point we need a few sort of settings first is what is the maximum received value you want to look at so when you do fixed point you should also fix the window, usually I take it as 3 or 4 you can play around with this and see it how it goes that is one and then the next one is the setting of the number of bits upto 4, but how many bits?

So one way in which I like to do this is to set something called maxint max of the maximum quantized value that you want to have so maybe we will call it max quantized r so you can have that as 31 for instance. So 31 means it is not going to exceed 31 at all the quantized received value so these are two settings that I use and then once you do that you have to now once r comes in and you have done the functioning for the message part, you have to do the quantization, so how do you do the quantization?

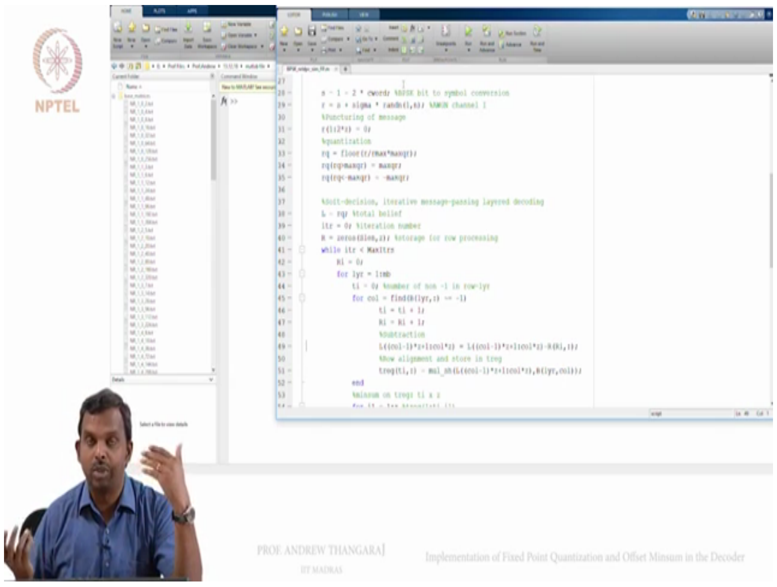
It is quite easy, so r quantized equal floor of r by r max multiplied by max qr.

So I am simply dividing by the maximum possible r max that I want to have, then multiplying by max qr and flooring so this will convert to integer and all that but then except that r may have gone greater than r max so what I do for that is rq of rq greater than r max equals r max and then rq of rq less than minus r max it can also go on the other direction and that I will set as minus (r max not r max I am sorry) max qr minus max qr, okay so this is the simple quantization and clipping also that is done.

So at the end of this I have integer values for rq and if it is greater than (I am sorry I should be careful here) it is also max qr, this is also minus max qr so if it has exceeded max qr of either side I simply set it to that value. So this gives me integer values the receiver and this L is going to be rq, okay so this is just one very quick way to deal with the quantization number 1, this will result in some sub optimal performance, so there will be lot of values which (()) (3:04) with real numbers but now they have become quantized so that is one thing and then after that there is really very little change that you have to do.

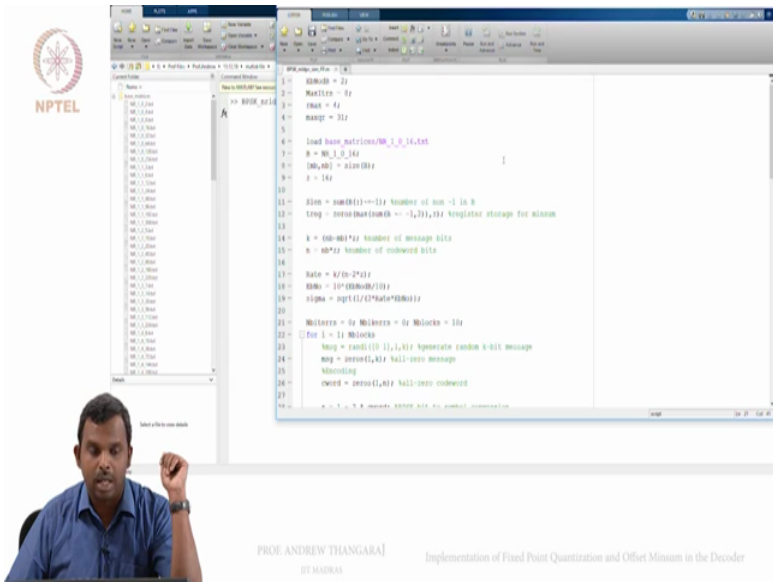
So few changes that you have to be careful about is first this L, okay so this L is set as rq now and then as you see in this iteration the value of L grows and then there is also this minus negative sign that I am doing for r and L so you have to be slightly careful when you quantize these things so that things do not blow up and still the calculation is done in a reasonable way, okay.

(Refer Slide Time: 3:48)



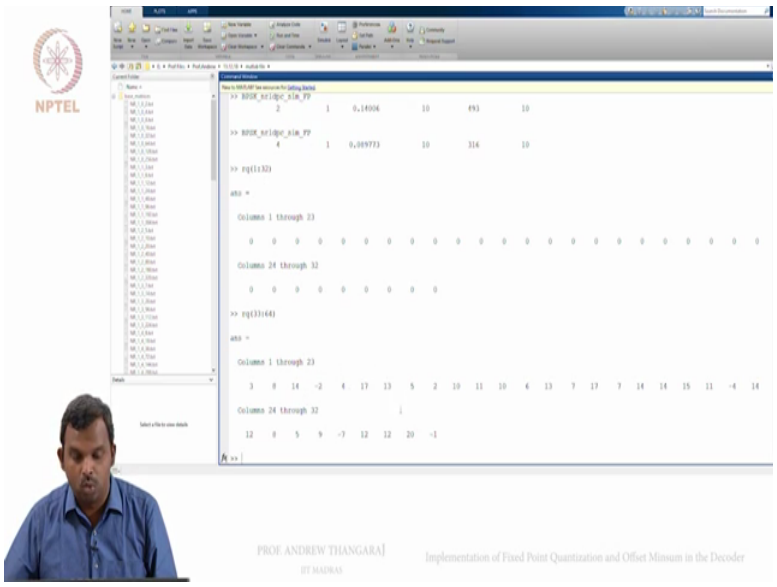
```
    s = 1 - 2 * cword; %BPSK bit to symbol conversion
    r = s + sigma * randn(1,n); %AWGN channel I
    %Puncturing of message
    r(1:2*z) = 0;
    %quantization
    rq = floor(r/rmax*maxqr);
    rq(rq>maxqr) = maxqr;
    rq(rq<-maxqr) = -maxqr;

    %Soft-decision, iterative message-passing layered decoding
    L = rq; %total belief
    itr = 0; %iteration number
    R = zeros(Slen,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
                    L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)-R(Ri,:);
                    %Row alignment and store in treg
                    treg(ti,:) = mul_sh(L((col-1)*z+1:col*z),B(lyr,col));
            end
            %minsum on treg: ti x z
```

PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder



```
EbNodB = 2;
MaxItrs = 8;
rmax = 4;
maxqr = 31;

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 = 10;
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
```

PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder



```
>> BPSK_nrldpc_sim_FP
           2             1      0.14006          10          493           10

>> BPSK_nrldpc_sim_FP
           4             1     0.009773          10          316           10

>> rq(1:32)

ans =

  Columns 1 through 23

    0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 24 through 32

    0     0     0     0     0     0     0     0     0

>> rq(33:64)

ans =

  Columns 1 through 23

    3     8    14    -2     4    17    13     5     2    10    11    10     6    13     7    17     7    14    14    15    11    -4    14

  Columns 24 through 32

   12     8     5     9    -7    12    12    20    -1

>>
```

PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

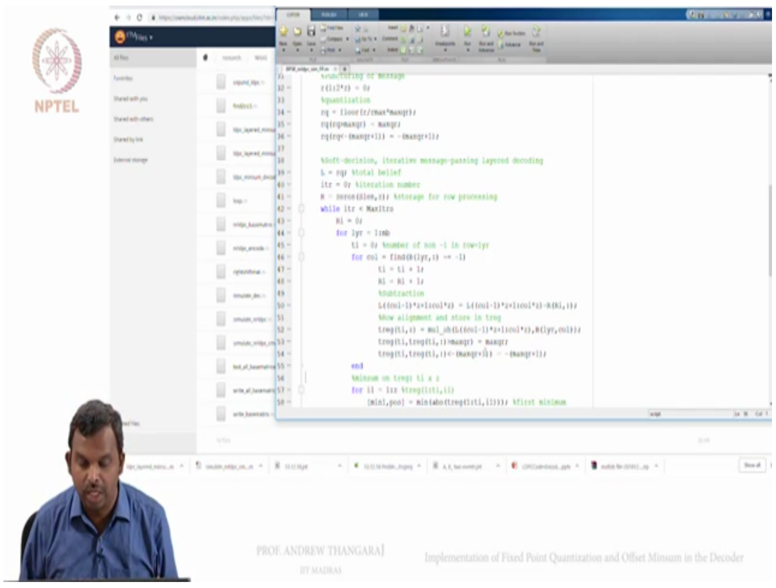
So some of the things one needs to pay attention to is how much will you let L grow and how much will you let r grow, okay so because remember even r is a value here so r is stored here, so you will have to keep limiting both L and r to some reasonable range, okay so one needs to pay some attention here so it is not just easy to quantize the r and let it run because the other values also you have to quantize you have to fix the bit in some sense, okay.

So maybe we should just try running this for a while to see if this is effected any of the performance I will run it for 10 blocks and see what happens? So you see you are getting some errors so maybe we can go to a larger SNR so we are getting a lot of errors here, so you see even at high SNR's the errors do not go down so the quantization seems to be hurting you in some way but one needs to be also be careful about how we have done the quantization, we just limited it but then we are not limiting the other values carefully so maybe it is good to check if rq happen properly or not?

So this will all be 0, okay so we will go to 33 colon 64, okay so it seems to be doing okay here the quantization is sort of worked okay maybe we need to do more iterations 8 is (too much) too less maybe we will go 20 iterations let us run 4db so we should see some so we are still getting 10 errors I am not very happy about it, okay alright so to account for the increasing the value of L in your inside your decoder so usually the L value so if you cut off the r and capital R and storage for the row at 31, so that is just 5 bits what we are using here.

So it is good to use more value here so usually most people use 2 bits more for the L value 127 and then one needs to look at the cut off closely.

(Refer Slide Time: 5:58)



```
    %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(Slen,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
                  L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)-R(Ri,:);
                  %Row alignment and store in treg
                  treg(ti,:) = mul_sh(L((col-1)*z+1:col*z),B(lyr,col));
                  treg(ti,treg(ti,:)>maxqr) = maxqr;
                  treg(ti,treg(ti,:)<-(maxqr+1)) = -(maxqr+1);
            end
            %minsum on treg: ti x z
            for i1 = 1:z %treg(1:ti,i1)
                [min1,pos] = min(abs(treg(1:ti,i1))); %first minimum
```

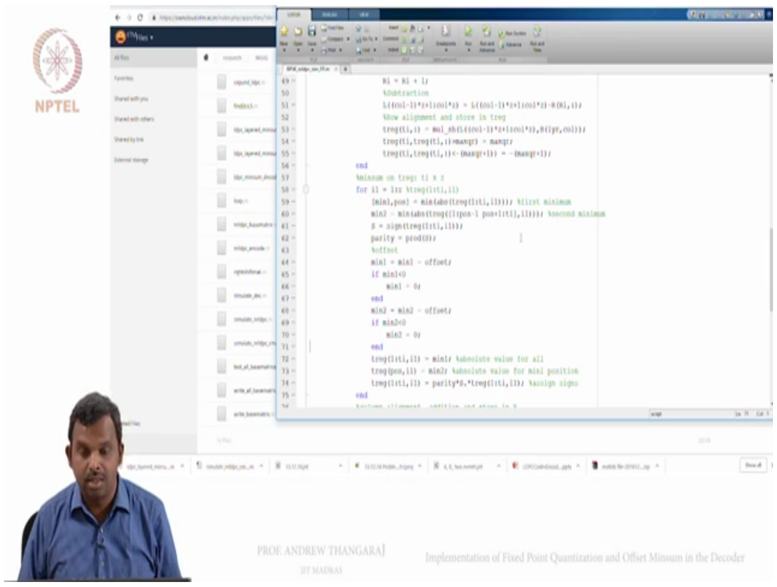
NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

So once again here so if you remember max qr was 31, so you can limit it to 31 so here I think if it is less than so usually since it is 2’s complement 1 cannot do this slight little trick here so that it goes upto minus 32 and plus 31, okay so that is the range for the rq here, L is rq but remember L is going to be allowed to grow upto 127, okay so minus 128 plus 127 so that is something that one needs to be careful about here so L subtraction is happening fine, but when this treg happens, okay so I do not want to store more than 5 bits here, okay so remember L was L I am going to allow it to grow upto 127, okay.

So when I do the subtraction after I subtract I have to further make sure that what I store in treg is only 5 bits long okay so I mean 6 bits total including 1 sign bit so it should go up only upto minus 31 to plus 31 so that is the adjustment that one needs to make.

So after I am done with this I should make sure that that happens so if one quick way of doing it is treg of ti comma treg of ti comma colon greater than max qr equals max qr. So if it exceeds you do that and then if it goes lesser less than if you remember minus max qr plus 1 then you set it as minus max qr plus 1, okay so this is the way in which I ensure that my treg value does not exceed the 6 bit limit minus 32 to plus 31 so that is where the range of treg lies.

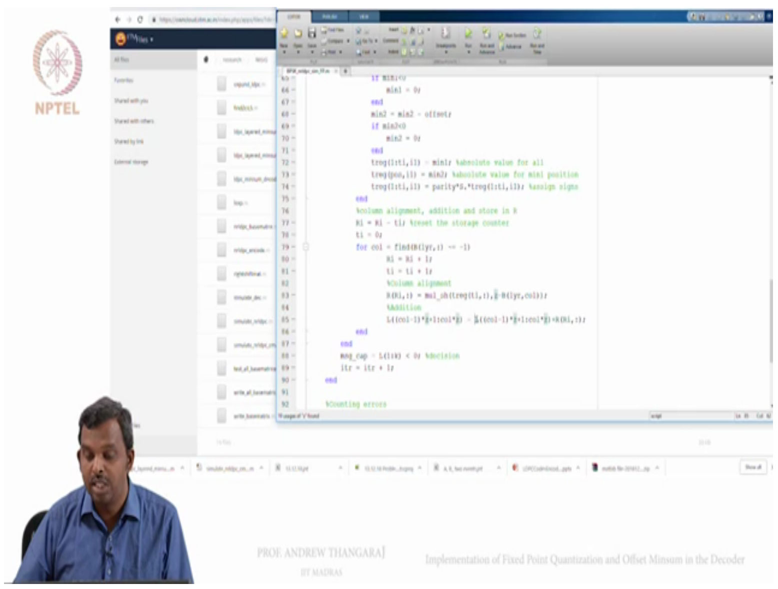
(Refer Slide Time: 8:02)



```
            Ri = Ri + 1;
            %Subtraction
            L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)-R(Ri,:);
            %Row alignment and store in treg
            treg(ti,:) = mul_sh(L((col-1)*z+1:col*z),H(lyr,col));
            treg(ti,treg(ti,:)>maxqr) = maxqr;
            treg(ti,treg(ti,:)<-(maxqr+1)) = -(maxqr+1);
        end
        %minsum on treg: ti x z
        for i1 = 1:z %treg(1:ti,i1)
          [min1,pos] = min(abs(treg(1:ti,i1))); %first minimum
          min2 = min(abs(treg([1:pos-1 pos+1:ti],i1))); %second minimum
          S = sign(treg(1:ti,i1));
          parity = prod(S);
          %offset
          min1 = min1 - offset;
          if min1<0
              min1 = 0;
          end
          min2 = min2 - offset;
          if min2<0
              min2 = 0;
          end
          treg(1:ti,i1) = min1; %absolute value for all
          treg(pos,i1) = min2; %absolute value for min1 position
          treg(1:ti,i1) = parity*S.*treg(1:ti,i1); %assign signs
        end
```

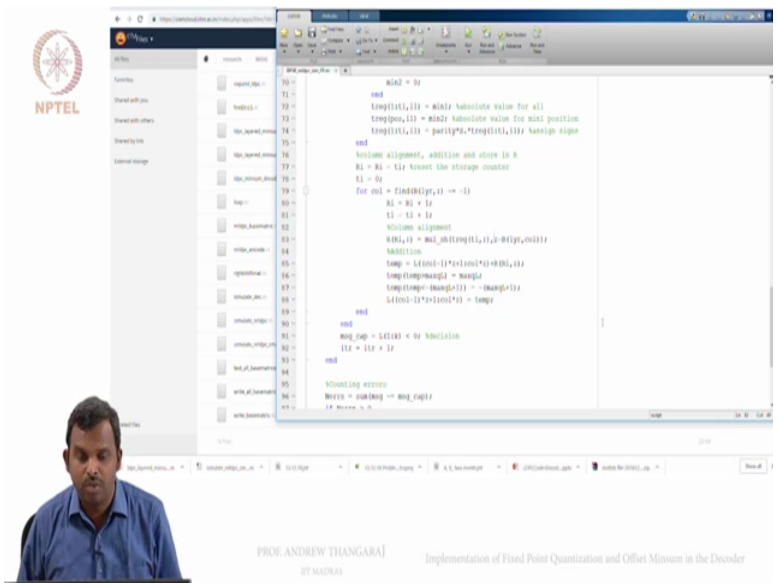
PROF. ANDREW THANGARAJ
IIT MADRAS

Implementation of Fixed Point Quantization and Offset Minsum in the Decoder



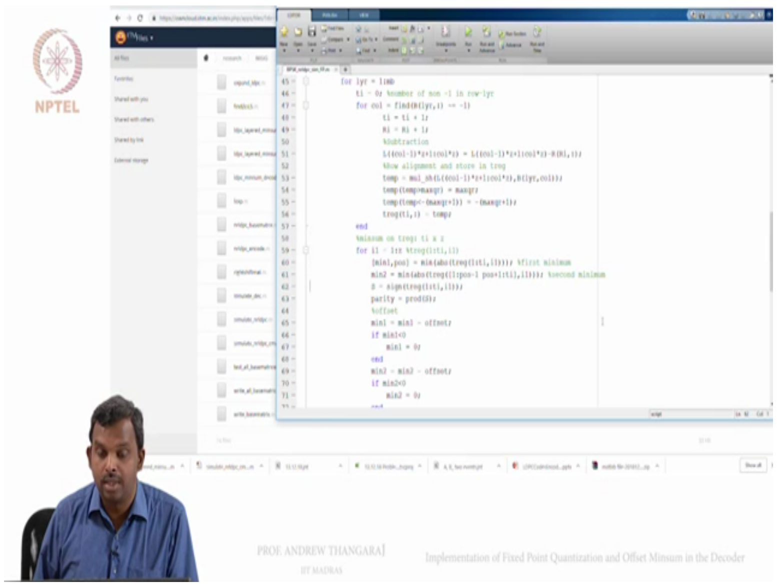
```
            if min1<0
                min1 = 0;
            end
            min2 = min2 - offset;
            if min2<0
                min2 = 0;
            end
            treq(1:ti,i1) = min1; %absolute value for all
            treq(pos,i1) = min2; %absolute value for min1 position
            treq(1:ti,i1) = parity*S.*treq(1:ti,i1); %assign signs
        end
        %column alignment, addition and store in R
        Ri = Ri - ti; %reset the storage counter
        ti = 0;
        for col = find(B(lyr,:) ~= -1)
            Ri = Ri + 1;
            ti = ti + 1;
            %Column alignment
            R(Ri,:) = mul_sh(treq(ti,:),z-B(lyr,col));
            %Addition
            L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)+R(Ri,:);
        end
    end
    msg_cap = L(1:k) < 0; %decision
    itr = itr + 1;
end

%Counting errors
```

PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder



```
                min2 = 0;
            end
            treq(1:ti,i1) = min1; %absolute value for all
            treq(pos,i1) = min2; %absolute value for min1 position
            treq(1:ti,i1) = parity*S.*treq(1:ti,i1); %assign signs
        end
        %column alignment, addition and store in R
        Ri = Ri - ti; %reset the storage counter
        ti = 0;
        for col = find(B(lyr,:) ~= -1)
            Ri = Ri + 1;
            ti = ti + 1;
            %Column alignment
            R(Ri,:) = mul_sh(treq(ti,:),z-B(lyr,col));
            %Addition
            temp = L((col-1)*z+1:col*z)+R(Ri,:);
            temp(temp>maxqL) = maxqL;
            temp(temp<-(maxqL+1)) = -(maxqL+1);
            L((col-1)*z+1:col*z) = temp;
        end
    end
    msg_cap = L(1:k) < 0; %decision
    itr = itr + 1;
end

%Counting errors
Nerrs = sum(msg ~= msg_cap);
```

PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder



```
    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
                L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)-R(Ri,:);
                %Row alignment and store in treq
                temp = mul_sh(L((col-1)*z+1:col*z),B(lyr,col));
                temp(temp>maxqr) = maxqr;
                temp(temp<-(maxqr+1)) = -(maxqr+1);
                treq(ti,:) = temp;
        end
        %minsum on treq: ti x z
        for i1 = 1:z %treq(1:ti,i1)
            [min1,pos] = min(abs(treq(1:ti,i1))); %first minimum
            min2 = min(abs(treq([1:pos-1 pos+1:ti],i1))); %second minimum
            S = sign(treq(1:ti,i1));
            parity = prod(S);
            %offset
            min1 = min1 - offset;
            if min1<0
                min1 = 0;
            end
            min2 = min2 - offset;
            if min2<0
                min2 = 0;
```

PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

After that I do this and then I want to do an offset minsum here, so maybe I will set up that offset also offset maybe we will set it as 2, okay so 2 maybe okay. So the offset it is convenient to set it in min, okay the min 1 and min 2 are the ones that need to be offset. So what you can do is if min 1 greater than offset okay so this is the check you have to do min 1 is min 1 minus offset, okay. So this you can end and then the same thing for min 2 as well.

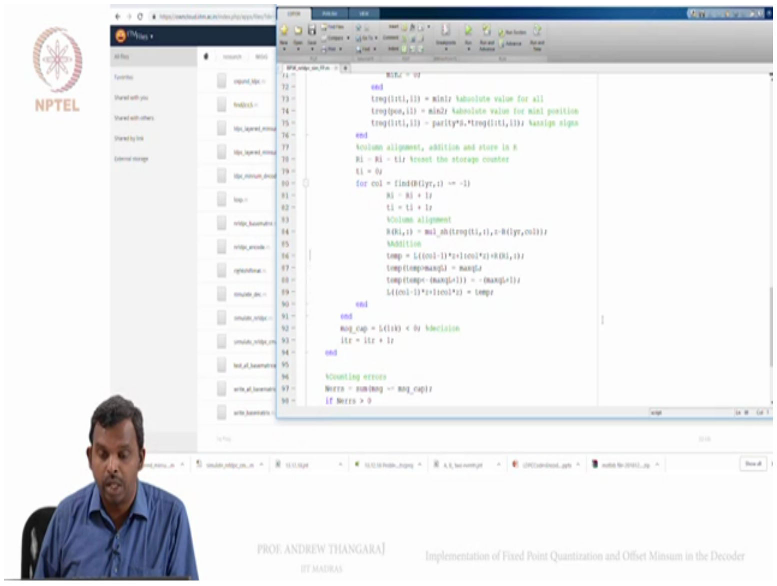
So slightly better way to do it is the following I am sorry so you do the subtraction first, okay and the you see if it is less than 0 and you set it to 0, okay so same thing for min 2 as well yeah that is better this is the cleaner way to implement the offset, okay. So like I said this is a way in which minsum can be made slightly more accurate so you introduce the offset here, you subtract the offset and if the min 1 is gone negative because of that you set it to 0 so same thing with min 2 and if min 2 has gone less than 0 you set min 2 to 0 that is it.

So after that nothing else changes as far as the parity is concerned, then when you add L here so this you have to check if because I am adding here so I am adding some value to L you have to check if it is gone greater than max ql okay so max ql was 127 I do not want L to exceed 127 either, okay so I am going to check if L has exceeded that and for that one needs to do the same sort of adjustment here which is a little bit more painful.

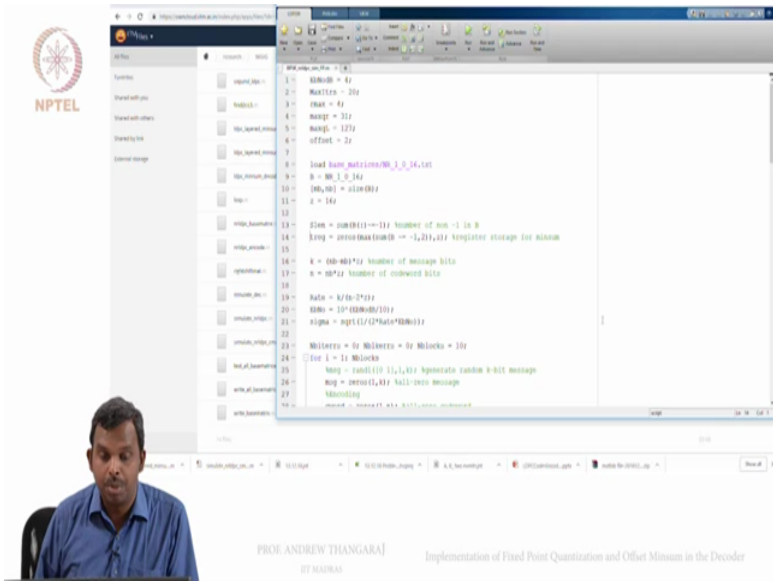
So what one can do here is the following, so instead of assigning it like this I will assign it to temp here and this I will take as temp add it up so this temporary storage will be slightly larger and then I do temp of temp greater than max ql to be equal to max ql and temp of temp less than minus max ql plus 1 to be equal to minus max ql plus 1 so this makes sure that the range of L is not violated and then I store it in temp I can do in fact something like this in treg here also instead of storing this in treg I can do temp here and simply do everything with temp and finally assign it to temp that is also I think is a good idea instead of doing this as if exchange here so let me just do that off carefully so I will assign it to temp and say temp of temp here this is better this looks cleaner code so that works out better for us and finally treg of will be ti comma colon equals temp okay so this is lot of slightly clean.

So I do the fix point manipulation with the temporary storage here some register something maybe slightly larger and then you finally quantize it down and store it in treg, okay so this is lot of clean.

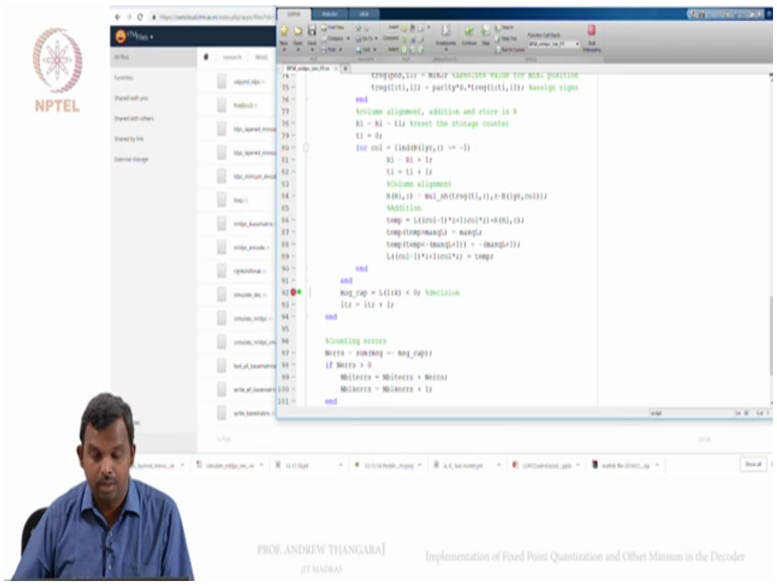
(Refer Slide Time: 12:00)


PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

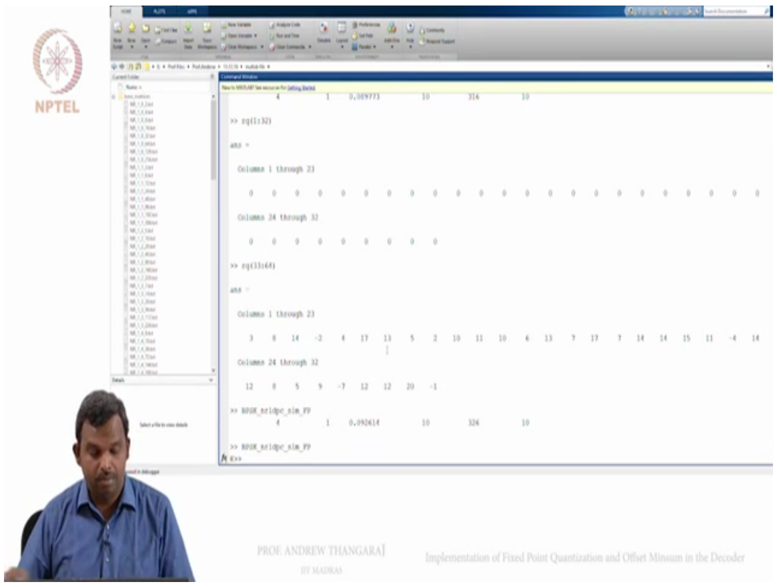
```
                min2 = 0;
            end
            treg(1:ti,l1) = min1; %absolute value for all
            treg(pos,l1) = min2; %absolute value for min1 position
            treg(1:ti,l1) = parity*S.*treg(1:ti,l1); %assign signs
        end
        %column alignment, addition and store in R
        Ri = Ri - ti; %reset the storage counter
        ti = 0;
        for col = find(B(lyr,:) ~= -1)
            Ri = Ri + 1;
            ti = ti + 1;
            %Column alignment
            R(Ri,:) = mul_sh(treg(ti,:),z-B(lyr,col));
            %Addition
            temp = L((col-1)*z+1:col*z)+R(Ri,:);
            temp(temp>maxqL) = maxqL;
            temp(temp<-(maxqL+1)) = -(maxqL+1);
            L((col-1)*z+1:col*z) = temp;
        end
    end
    msg_cap = L(1:k) < 0; %decision
    itr = itr + 1;
end

%Counting errors
Nerrs = sum(msg ~= msg_cap);
if Nerrs > 0
```


PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

```
EbNodB = 4;
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 = 10;
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
```


PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

```
            treg(pos,l1) = min2; %absolute value for min1 position
            treg(1:ti,l1) = parity*S.*treg(1:ti,l1); %assign signs
        end
        %column alignment, addition and store in R
        Ri = Ri - ti; %reset the storage counter
        ti = 0;
        for col = find(B(lyr,:) ~= -1)
            Ri = Ri + 1;
            ti = ti + 1;
            %Column alignment
            R(Ri,:) = mul_sh(treg(ti,:),z-B(lyr,col));
            %Addition
            temp = L((col-1)*z+1:col*z)+R(Ri,:);
            temp(temp>maxqL) = maxqL;
            temp(temp<-(maxqL+1)) = -(maxqL+1);
            L((col-1)*z+1:col*z) = temp;
        end
    end
    msg_cap = L(1:k) < 0; %decision
    itr = itr + 1;
end

%Counting errors
Nerrs = sum(msg ~= msg_cap);
if Nerrs > 0
    Nbiterrs = Nbiterrs + Nerrs;
    Nblkerrs = Nblkerrs + 1;
end
```



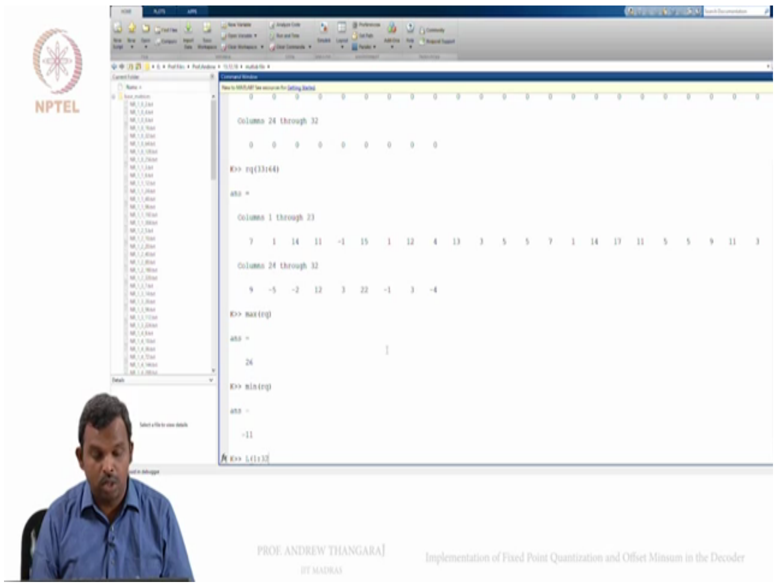
```
           4           1    0.089773          10         314          10

>> rq(1:32)

ans =

  Columns 1 through 23

     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 24 through 32

     0     0     0     0     0     0     0     0     0

>> rq(33:64)

ans =

  Columns 1 through 23

     3     8    14    -2     4    17    13     5     2    10    11    10     6    13     7    17     7    14    14    15    11    -4    14

  Columns 24 through 32

    12     8     5     9    -7    12    12    20    -1

>> BPSK_arldpc_sim_FP
           4           1    0.092614          10         326          10

>> BPSK_arldpc_sim_FP
K>>
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

Same thing I did here for L as well I first did the addition into some temporary thing so upto now it could be little bit bigger and then I got it down and then store it in temporary, okay so other than that I do not think you need to do any other change this should work just like that 20 iterations maybe that is a good idea, so maybe it is good to check few iterations ad see how this value works out, okay.

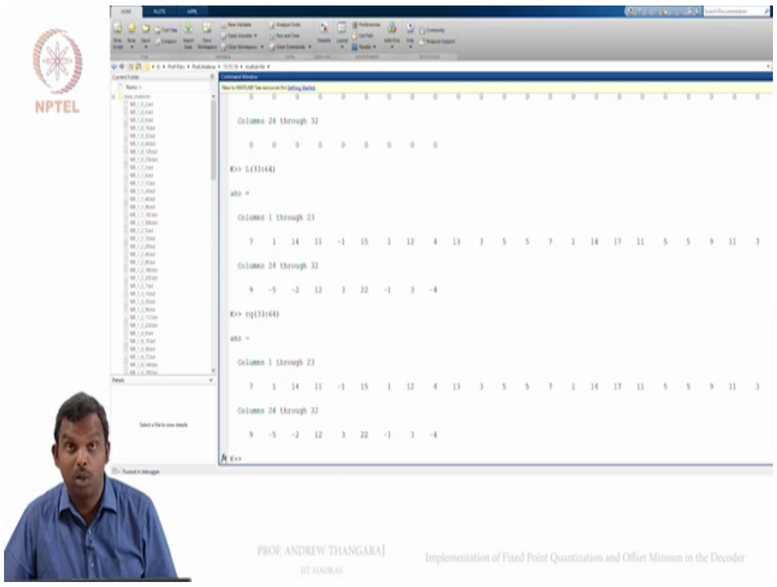
So maybe I will stop somewhere here this is a good place to stop, I will set up a break point and then let it run, see if this is running okay it seems to be running maybe we will clear the whole thing up.

(Refer Slide Time: 12:44)



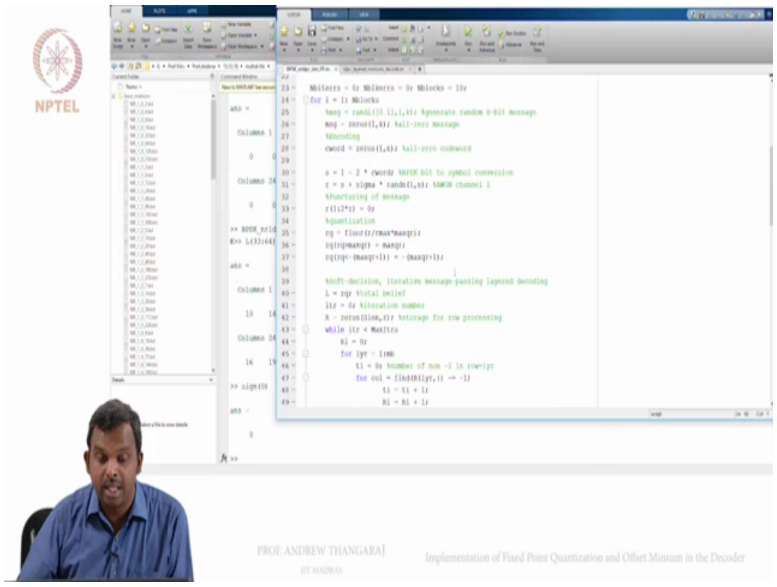
```
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 24 through 32

     0     0     0     0     0     0     0     0     0

K>> rq(33:64)

ans =

  Columns 1 through 23

     7     1    14    11    -1    15     1    12     4    13     3     5     5     7     1    14    17    11     5     5     9    11     3

  Columns 24 through 32

     9    -5    -2    12     3    22    -1     3    -4

K>> max(rq)

ans =

    26

K>> min(rq)

ans =

   -11

K>> L(1:32
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder



```
  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0

  Columns 24 through 32

  0    0    0    0    0    0    0    0    0

K>> L(33:64)

ans =

  Columns 1 through 23

  7    1   14   11   -1   15    1   12    4   13    3    5    5    7    1   14   17   11    5    5    9   11    3

  Columns 24 through 32

  9   -5   -2   12    3   22   -1    3   -4

K>> rq(33:64)

ans =

  Columns 1 through 23

  7    1   14   11   -1   15    1   12    4   13    3    5    5    7    1   14   17   11    5    5    9   11    3

  Columns 24 through 32

  9   -5   -2   12    3   22   -1    3   -4

K>>
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder



```
Nbiterrs = 0; Nblkerrs = 0; Nblocks = 10;
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(Slen,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;
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

So let us see rq of 1 colon 32 this will be all 0 because I am puncturing this, so if you look at 33 colon 64 the next two blocks you see a lot of values and it goes from (minus 31 to plus) minus 32 to plus 31 the way I have seen it one can do max of rq just to check it has gone upto 26 min of rq upto minus 11, so that is the range of max so you expect more positive values then negative values that is good, okay.

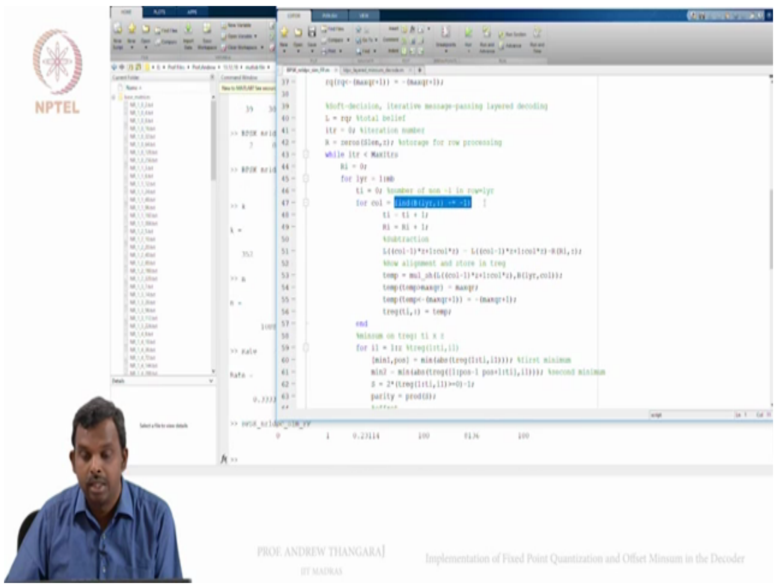
So now what about L? So if you see actually L of 1 colon 32 in the first iteration it will still be 0, nothing would have been corrected in the first iteration because it got assigned as 0, nothing really flows you will see in the third iteration onwards you will start seeing some values here but if you see L of 33 colon 64 something will be different, so you can see rq we saw this just before, okay.

So is everything exactly the same it is not very good news so maybe okay looks to be equal for now so maybe that is the first iteration and that block does not get corrected let us say (()) (13:56) next iteration, continue let us try L, still not changing, next iteration hopefully something should start flowing now. One important thing about functioning is when you said r to be equal to 0, your rq will also end up being 0 but then MATLAB's sin function treats 0 as 0.

So for instance if you look at what MATLAB sin function does it does sin of 0it gives a 0, so that is actually not a very good way in my opinion to implement the sin function so when you do that and when you do product of sin everything will go for a toss. So better thing to do is to do something like this to implement your own sin function and put a greater than equal to 0 here, okay.

So if you do this then things work out quite okay. So the sins instead of relying on MATLAB sin function we do 2 into the value you want greater than equal to 0. So if it is equal to 0 I am going to assign plus 1 in the sin, so this is an important change that one needs to make to make sure things work particularly in the punctured case and the quantized case when you may have things equal to 0 otherwise everything will keep on being 0 all the time, none of the rows will effectually effectively be there, okay so this is one of the bugs that you need to correct when you quantize, okay.

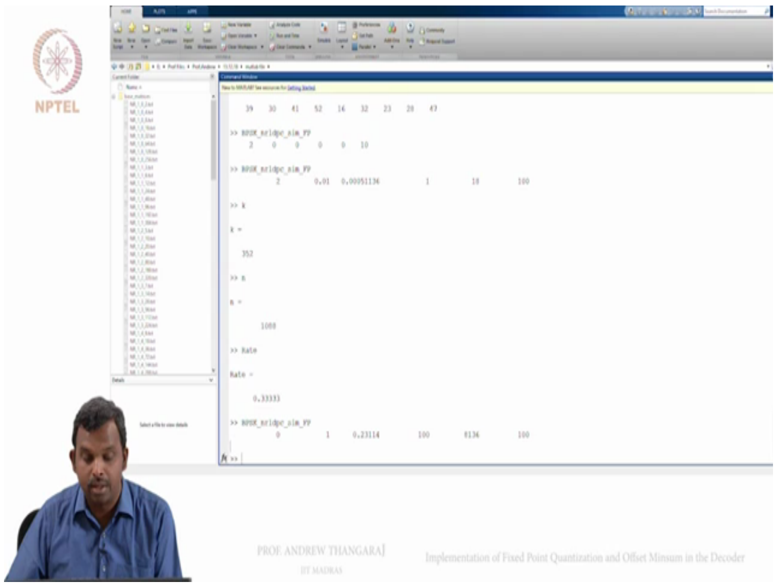
(Refer Slide Time: 15:45)



```
rq(rq<-(maxqr+1)) = -(maxqr+1);

%soft-decision, iterative message-passing layered decoding
L = rq; %total belief
itr = 0; %iteration number
R = zeros(Slen,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
                L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)-R(Ri,:);
                %row alignment and store in treg
                temp = mul_sh(L((col-1)*z+1:col*z),B(lyr,col));
                temp(temp>maxqr) = maxqr;
                temp(temp<-(maxqr+1)) = -(maxqr+1);
                treg(ti,:) = temp;
        end
        %minsum on treg: ti x z
        for i1 = 1:z %treg(1:ti,i1)
            [min1,pos] = min(abs(treg(1:ti,i1))); %first minimum
            min2 = min(abs(treg([1:pos-1 pos+1:ti],i1))); %second minimum
            S = 2*(treg(1:ti,i1)>=0)-1;
            parity = prod(S);
```

```
0       1     0.23114       100       8136       100
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder



```
   39   30   41   52   16   32   23   28   47

>> BPSK_nrldpc_sim_FP
    2    0    0    0    0   10

>> BPSK_nrldpc_sim_FP
           2        0.01   0.00051136          1          18          100

>> k

k =

   352

>> n

n =

        1088

>> Rate

Rate =

     0.33333

>> BPSK_nrldpc_sim_FP
           0           1      0.23114        100        8136          100

>> 
```

NPTEL
PROF. ANDREW THANGARAJ
IIT MADRAS
Implementation of Fixed Point Quantization and Offset Minsum in the Decoder

So I think with this most of the things are done, so let me just ensure that one little detail have taken care of carefully I think this looks okay to me so let us run this we will keep a break point here and see if everything is working correctly or not and then we can run this, okay so once again running (())(15:55) so let us run.

So if you run this you can see rq which is the quantized r from one colon 32 is 0 and rq from 33 colon 64 is some value there are some negatives here minus 5 1 but other values are okay and if you look at L of 33 colon 64 you should see some change here some of the values have gone up the minus has become plus, the minus 1 has become 0 so some in the positive direction but interestingly if you look at L of 1 colon 32 okay so this has some values as well so those values also been corrected and you are getting some changes here so this is what will happen when you do it and if you continue the iteration the L will change again remember this was quantized so this all becoming very very large, okay.

And if you look at this value also this is also becoming very very large and quickly one is converging to the correct answer, okay. So this is something that one can see maybe I can quit the debugging here and clear the break point then run it for some time and see how it (()) (17:03) so maybe we will run it later 2db db over n not okay this is the rate 1 by 3 is this quantized? It is quite a flow block length let us run it and see what happens?

Okay, so no errors no errors at 0db and then maybe I will run it for 100 blocks just to be sure sorry 2db over n not rate was 1 by 3, this is the quantized decoder and you can see even the quantized decoder works very well and gives you no errors 2db db over n not, okay. So hopefully this is clean now I have not made any error change, this is a clean decoder. So you can see the offset also plays a very good role in improving the error rate.

So 1 out of 100 was an error at (())(17:54) over n not of 2db you can even reduce the db over n not if you like, I remove this is just the block length of 1000, so if you look at k n which is 1000, rate will be 1 by 3, okay so because I did that and so this is easy enough to do you can even reduce the bit error rate the eb over n not further and you will see more errors will start creeping in so this point (())(18:24) 10 power minus 2 1 out of 100 were an error, if you go to 0db you will get more errors at these block lengths and the whole decoder is working sort of reasonably, we have been cooperated most details that need to be there the puncturing of the first part, we did not incorporate the shortening and the puncturing of the parity that I will like I said will leave as an exercise for you to make a modification it is not too difficult the changes have to be done here and there so here you can see the (())(18:53) 1 all the things were an error. So 0db over n not is too low 2db is pretty good and 3 I am sure will be even better for these kind of codes, okay.

So that is the end of this lecture on LDPC codes and with this we come to the end of the LDPC codes part of this course so like I promise to the beginning of the course you have encoder code in the 5G standard and you have a decoder also sort of the working decoder I am sure it can be made better there are lots of bells and vessels you can add, but it is a pretty good decoder it is got quantized received values, it is running offset minsum and you can play around with these parameters to see how you can change and improve etc and some of this we will leave as exercises for you, okay so thank you very much this also concludes the second week of lectures and we will proceed to polar codes in the next week onwards, thank you very much.