Polar Codes

Using MATLAB

Content

- Intro To Polar Codes
- Polarization Transformation Encoder
- Successive Cancellation (SC) Decoder
- The MATLAB implementation Code
- MATLAB Results

Intro To Polar Codes

Polar codes break the wheel somewhat in the field of channel coding.

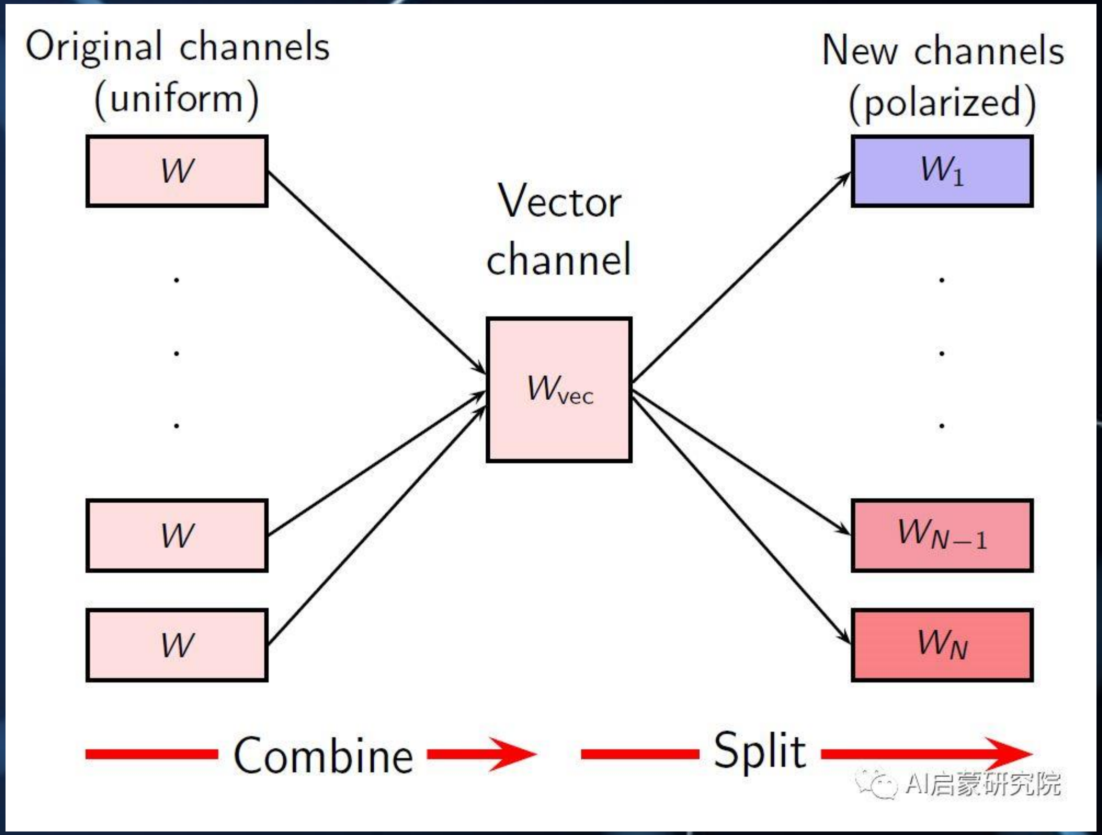
Polar codes operate on blocks of symbols/bits and are therefore technically members of the block code family.

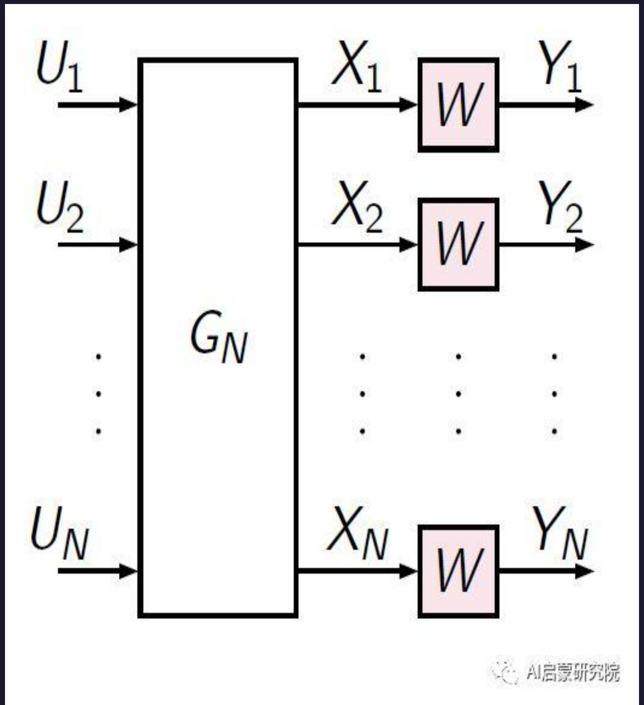
The transformation involves two key operations called channel combining and channel splitting.

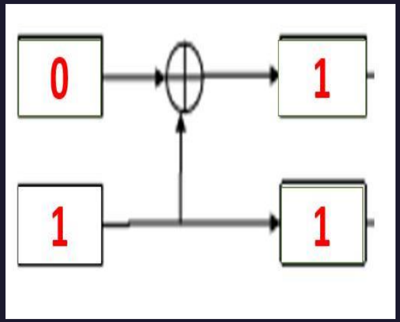
The real magic in polar codes lies in the clever bit manipulations and mappings to the channels at the encoder to convert a block of bits into a polarized bit stream at the receiver.

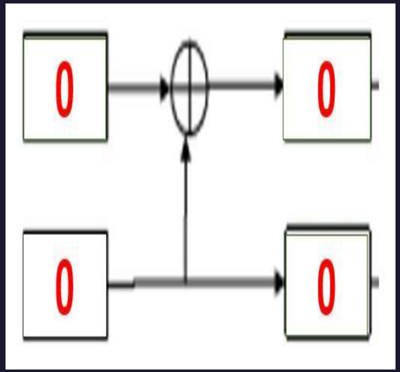
That is a received bit and its associated channel ends up being either a "good channel" or "bad channel" pole/category.

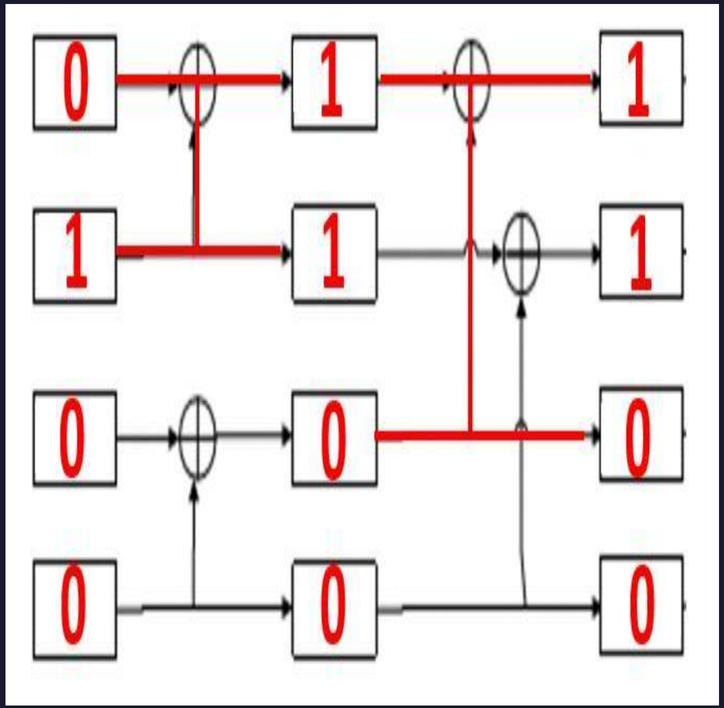
As the size of the bit block increases, the received bit stream polarizes in a way that the number of "good channels" approaches Shannon capacity. This phenomenon is what gives polar codes their name.

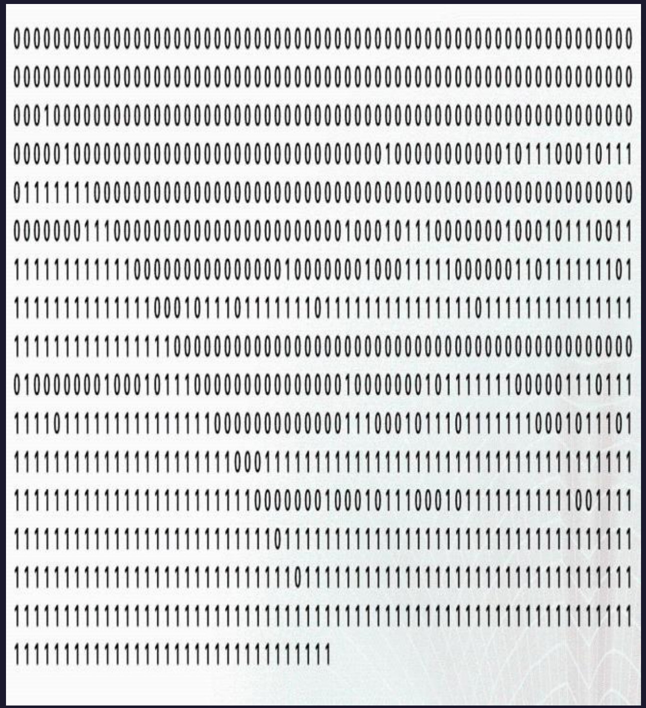


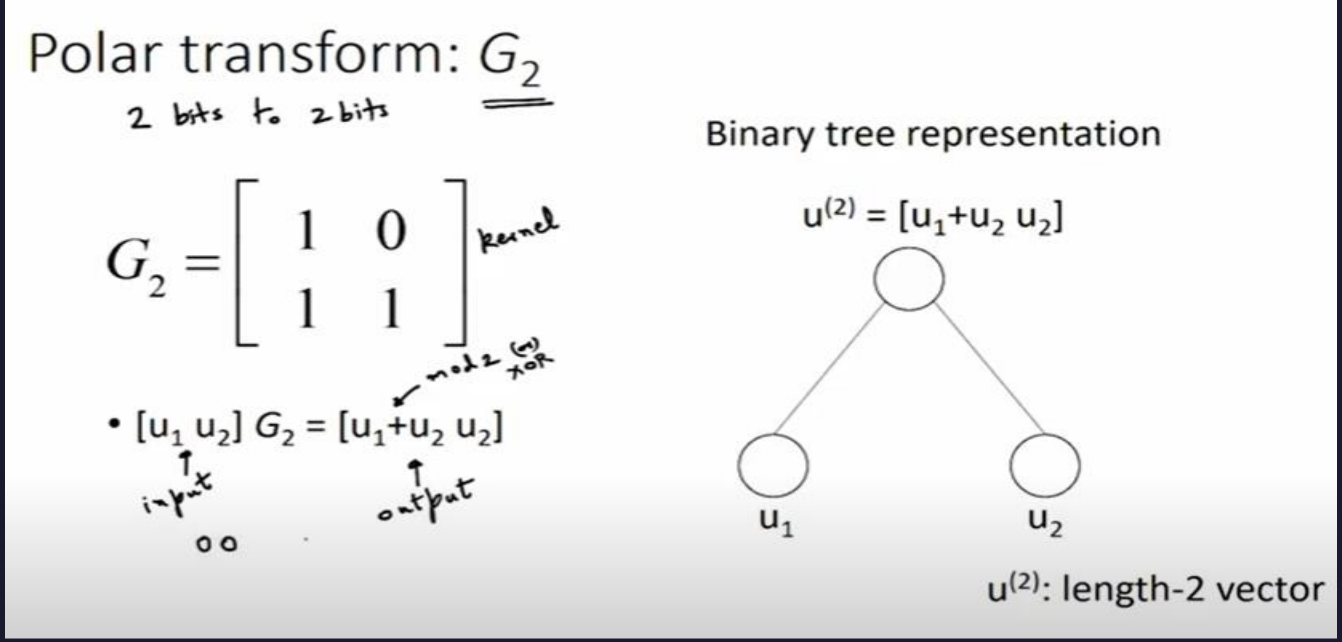












Polarization Transformation Encoder

Polar Transform: General

$$G_{2^n} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n}$$

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

The mapping $u_1^4 \mapsto x_1^4$ from the input of W_4 to the input of W^4 can be written as $x_1^4 = u_1^4 G_4$ where G_N is the generator matrix. Hence, we obtain the relation $W_4(y_1^4|u_1^4) = W^4(y_1^4|u_1^4 G_4)$ between the two transition probabilities.

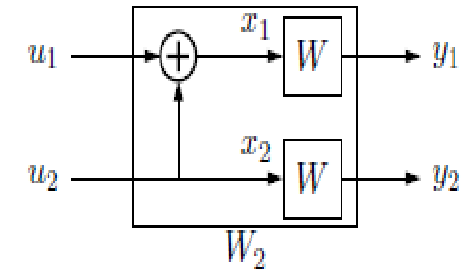
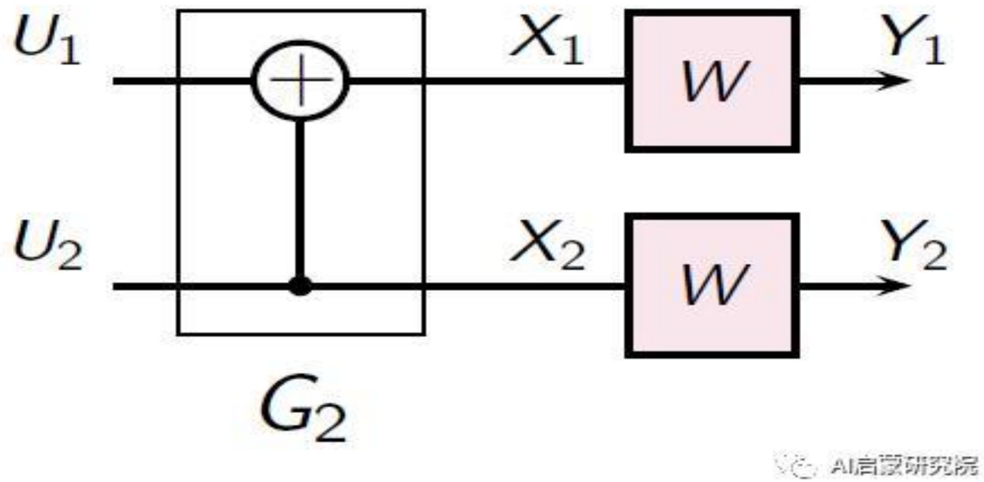


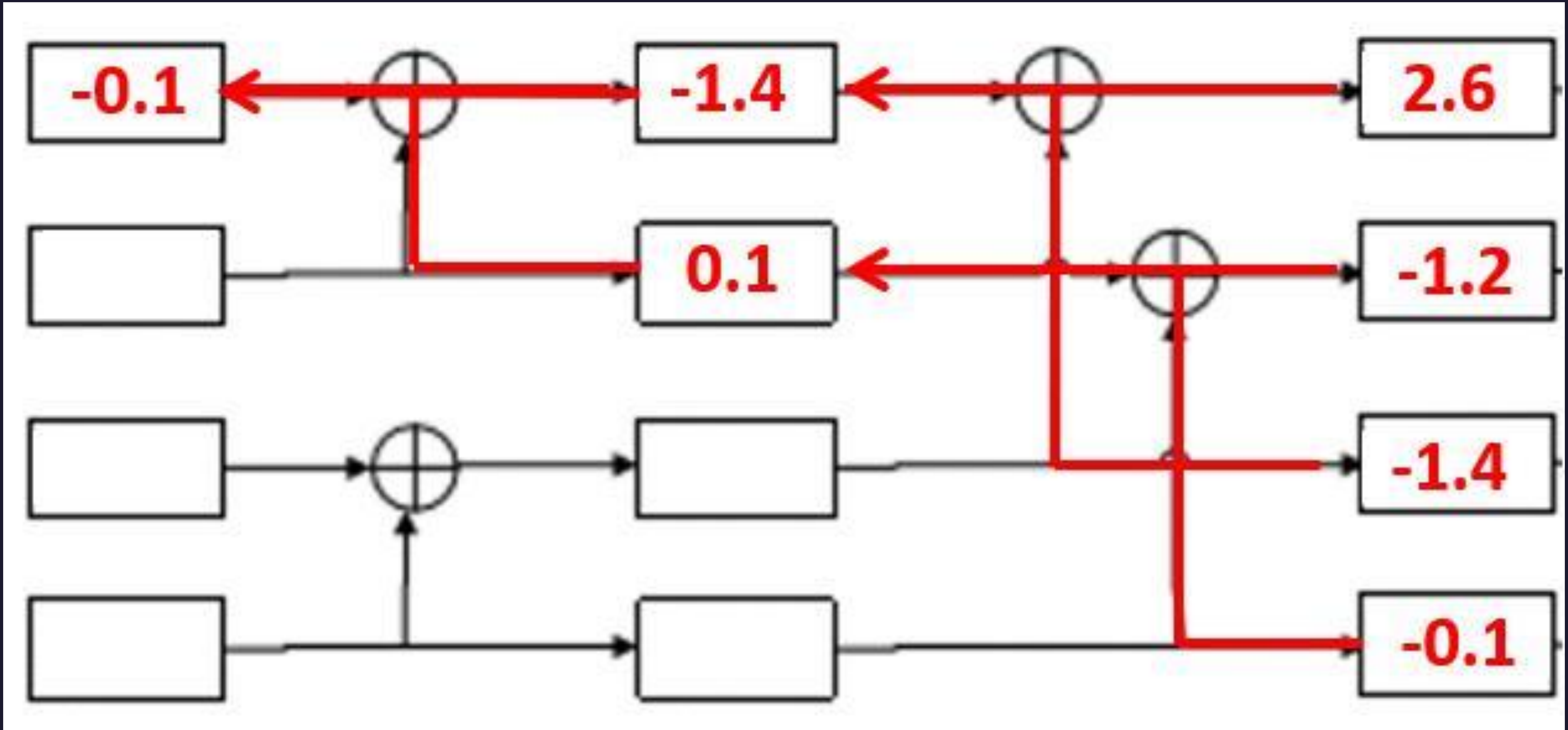
Figure 2.1: The channel W_2.



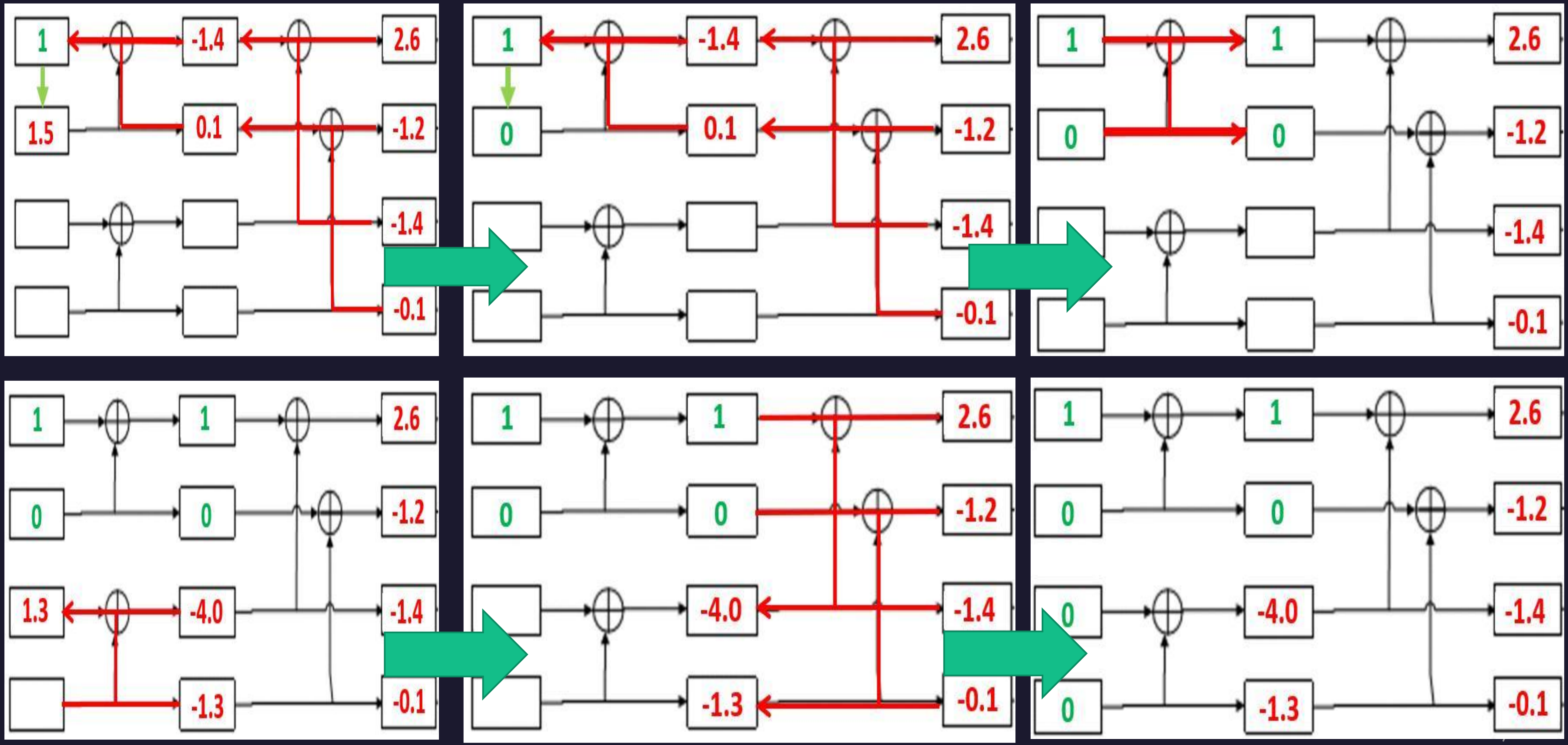
$$I(W) \triangleq I(X;Y) = \sum_{y\in\mathcal{Y}} \sum_{x\in\mathcal{X}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)}, \tag{2.1}$$

where X and Y are two discrete random variables corresponding to input and output, respectively, and $W(y|x)$ is the channel transition probability for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

Successive Cancellation (SC) Decoder



Successive Cancellation (SC) Decoder



The MATLAB implementation Code

Editor - D:\ASMAA 2021\collage\3rd\1. Intro to Digital Commu\4. labs\1. this year\lab2\part 3\polar\BSC with p (this is the one)\polar_code_main.m

polar_code_main.m

```matlab
clc
clear

N=8;                            % N Channels
n=log2(N);                      % N=2^n
A=[4 6 7 8];                    % information bit == message
u_A=[1 1 0 1];                  % information vetor == mostly it is equal to the message positions
AC=[1 2 3 5];                   % frozen bit == reliability sequance for N ==or it's called Q
u_AC=[0 0 0 0];                 % frozen vector ==the principal idea of polar coding is putting the
                                % information bits on those "good" channels whose capacity tend to be 1
                                % and the frozen bits on the "bad" ones.
snr=6;
p_vec = 0:0.01:0.5;                 % prob of BSC flipping
BER_vec  = zeros(size(p_vec));
%% Polar codes encoder
F=[1 0;1 1];                        %   F==G polar transformation kernal matrix
F_n=F;
    for i=1:(n-1)                   %   num of bits combined
                                    %   this is the polar transformation because G = kron( A,B )
                                    %   returns the Kronecker tensor product of matrices A and B .
        F_n=kron(F_n,F);            %   If A is an m -by- n matrix and B is a p -by- q matrix,
                                    %   then kron(A,B) is an m*p -by- n*q matrix formed by taking all possible
                                    %   products between the elements of A and the matrix B .
    end
I=eye(2^n);                          %   where eye(N) is the N-by-N identity matrix.
G_n=F_n;                             %   F==G polar transformation kernal matrix
u=u_A*I(A,:)+u_AC*I(AC,:);           %   u is the unencoded codeword to be compared with the received or decoded co
x=mod(u_A*G_n(A,:)+u_AC*G_n(AC,:),2); % x is the encoded codeword
```

Editor - D:\ASMAA 2021\collage\3rd\1. Intro to Digital Commu\4. labs\1. this year\lab2\part 3\polar\BSC with p (this is the one)\polar_code_main.m

polar_code_main.m

```matlab
%% polar_code_channel
%BSC with p prob of flipping
for p_index = 1:length(p_vec)

    x = BSChannel(x,p_vec(p_index));
    y=zeros(1,N);

    % mapping
    for i=1:N
        if(x(i)==0)
            y(i) = 1; % 0-->+1
        else
            y(i) = -1; % 1-->-1
        end
    end
    y =awgn(y,snr);
 %% Decoder
u_e = polar_code_SC_decoder(n,N,y,AC); %  u_e: estimated codeword
 %% computing BER
    count=0;
    for i=1:length(u)                   %u is the un-encoded bit seq using polarization transformation encoding

        if(u_e(i)~=u(i))
           count=count+1;
        end
    end
    BER_vec(p_index)=count/length(u);

```

The MATLAB implementation Code

Editor - D:\ASMAA 2021\collage\3rd\1. Intro to Digital Commu\4. labs\1. this year\lab2\part 3\polar\BSC with p (this is the one)\BSChannel.m

polar_code_main.m | polar_code_SC_decoder.m | BSChannel.m

```matlab
function rec_encoded_seq  = BSChannel(encoded_seq,p)
%
% Inputs:
%   encoded_seq:     The input  sequence to the channel
%   p:               The bit flipping probability
% Outputs:
%   rec_encoded_seq: The sequence after passing through the channel
%
% This function takes the encoded sequence passing through the channel, and
% generates the output sequence
encoded_seq      = ~~encoded_seq;
rec_encoded_seq  = zeros(size(encoded_seq));
rec_encoded_seq  = ~~rec_encoded_seq;

channel_effect = rand(size(rec_encoded_seq))<=p;

rec_encoded_seq = xor(encoded_seq,channel_effect);
rec_encoded_seq = rec_encoded_seq + 0;
```

Editor - D:\ASMAA 2021\collage\3rd\1. Intro to Digital Commu\4. labs\1. this ye

polar_code_main.m | polar_code_SC_decoder.m | BSChannel.m

```matlab
function [u_e] = polar_code_SC_decoder(n,N,y,AC)
%   n: number of level
%   N: length of codeword
%   y: undecoded codeword
%   AC: frozen bit
%   stage: current processing stage
%   LLR: log-likelihood ratio
%   HB: estimated hard bits
%   u_e: estimate of codeword

    %   initializing the log-likelihood ratio
    LLR = zeros([N,n+1]);
    LLR(:,1) = y;

    %polar_code_initializing_HardBit
    HardBits = zeros(N,n+1);
    for i=1:N
        if(LLR(i,1)>=0)
            HardBits(i,1) = 0;
        else
            HardBits(i,1) = 1;
        end
    end


    for stage=1:n
        % Calculate alpha_left
        % polar_code_updateLLR_Left
        Ns = 2^(n-stage+1);
```

- log-likelihood ratio (LLR) where

$$L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) = \ln\left(\frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i=0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1}|u_i=1)}\right);$$

The MATLAB implementation Code

polar_code_main.m | polar_code_SC_decoder.m | BSChannel.m

```matlab
    for stage=1:n
        % Calculate alpha_left
        % polar_code_updateLLR_Left
        Ns = 2^(n-stage+1);
        for j=1:Ns:N
            for i=j:j+Ns/2-1
                p = LLR(i,stage); %alpha_i
                q = LLR(i+Ns/2,stage); %alpha_(i+Ns/2)
                LLR(i,stage+1) = sign(p)*sign(q)*min([abs(p),abs(q)]);
            end
        end


        %polar_code_updateHB_L
        %Calculate beta_left
        Ns = 2^(n-stage+1);
        for j=1:Ns:N
            for i=j:j+Ns/2-1
                if(stage==n&&ismember(i,AC))
                    HardBits(i,stage+1) = 0;
                elseif(LLR(i,stage+1)>=0)
                    HardBits(i,stage+1) = 0;
                else
                    HardBits(i,stage+1) = 1;
                end
            end
        end

```

Editor - D:\ASMAA 2021\collage\3rd\1. Intro to Digital Commu\4. labs\1. this year\lab2\part 3\polar\BSC with p (this is the one)\polar_code_SC_decoder.m

polar_code_main.m | polar_code_SC_decoder.m | BSChannel.m

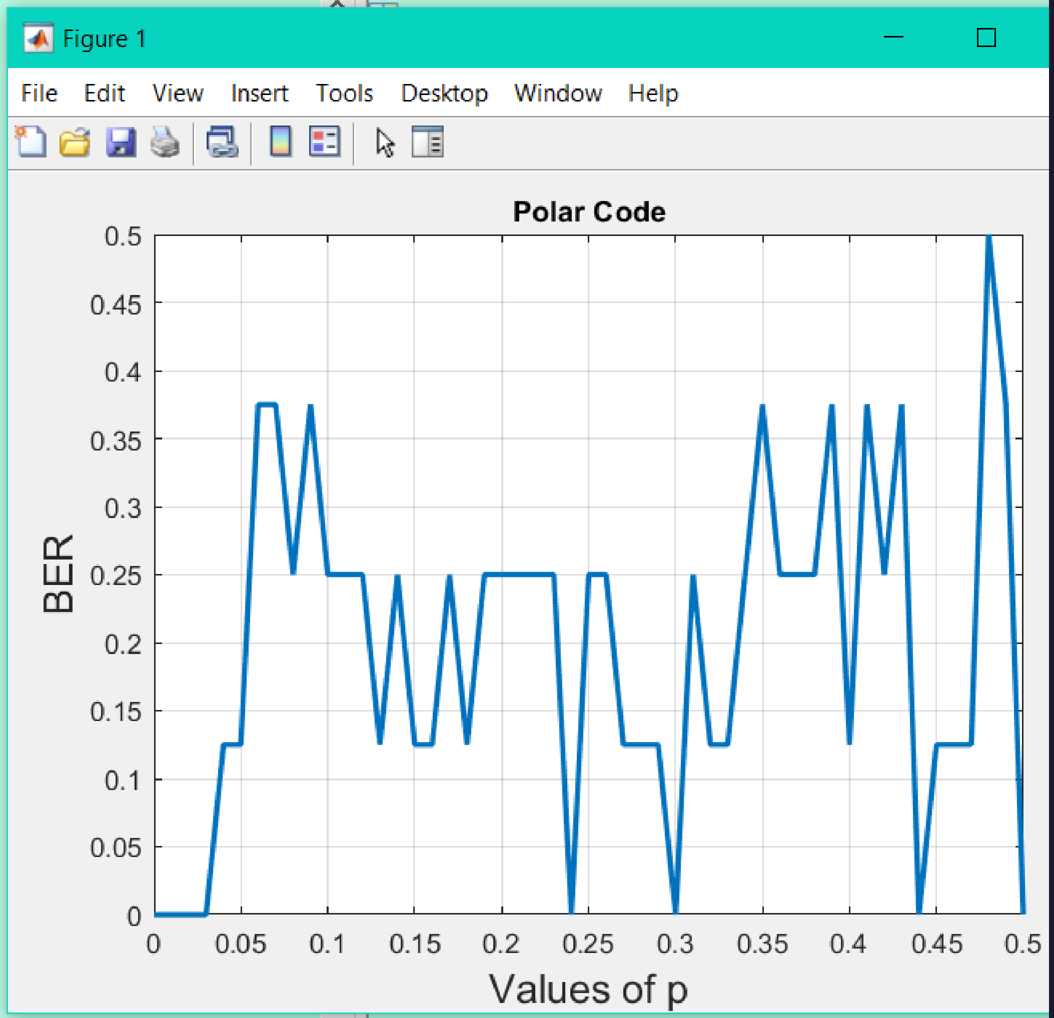
```matlab

        %polar_code_updateLLR_R
        % Calculate alpha_right
        Ns = 2^(n-stage+1);
        for j=1:Ns:N
            for i=j:j+Ns/2-1
                p = LLR(i,stage); %alpha_i
                q = LLR(i+Ns/2,stage); %alpha_(i+Ns/2)
                LLR(i+Ns/2,stage+1) = (1-2*HardBits(i,stage+1))*p+q;
            end
        end

        %HardBits = polar_code_updateHB_R
        %Calculate beta_right
        Ns = 2^(n-stage+1);
        for j=1:Ns:N
            for i=j+Ns/2:j+Ns-1
                if(stage==n&&ismember(i,AC))
                    HardBits(i,stage+1) = 0;
                elseif(LLR(i,stage+1)>=0)
                    HardBits(i,stage+1) = 0;
                else
                    HardBits(i,stage+1) = 1;
                end
            end
        end
    end
    u_e = HardBits(:,n+1)';
end
```

Results

```
Correct! your BER=0 when P=0
Correct! your BER=0 when P=1.000000e-02
Correct! your BER=0 when P=2.000000e-02
Correct! your BER=0 when P=3.000000e-02
some bits flipped & needed error correction,your BER=1.250000e-01 when P=4.000000e-02
some bits flipped & needed error correction,your BER=1.250000e-01 when P=5.000000e-02
some bits flipped & needed error correction,your BER=3.750000e-01 when P=6.000000e-02
some bits flipped & needed error correction,your BER=3.750000e-01 when P=7.000000e-02
some bits flipped & needed error correction,your BER=2.500000e-01 when P=8.000000e-02
some bits flipped & needed error correction,your BER=3.750000e-01 when P=9.000000e-02
some bits flipped & needed error correction,your BER=2.500000e-01 when P=1.000000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=1.100000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=1.200000e-01
some bits flipped & needed error correction,your BER=1.250000e-01 when P=1.300000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=1.400000e-01
some bits flipped & needed error correction,your BER=1.250000e-01 when P=1.500000e-01
some bits flipped & needed error correction,your BER=1.250000e-01 when P=1.600000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=1.700000e-01
some bits flipped & needed error correction,your BER=1.250000e-01 when P=1.800000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=1.900000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=2.000000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=2.100000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=2.200000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=2.300000e-01
Correct! your BER=0 when P=2.400000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=2.500000e-01
some bits flipped & needed error correction,your BER=2.500000e-01 when P=2.600000e-01
```



References

1. Book:

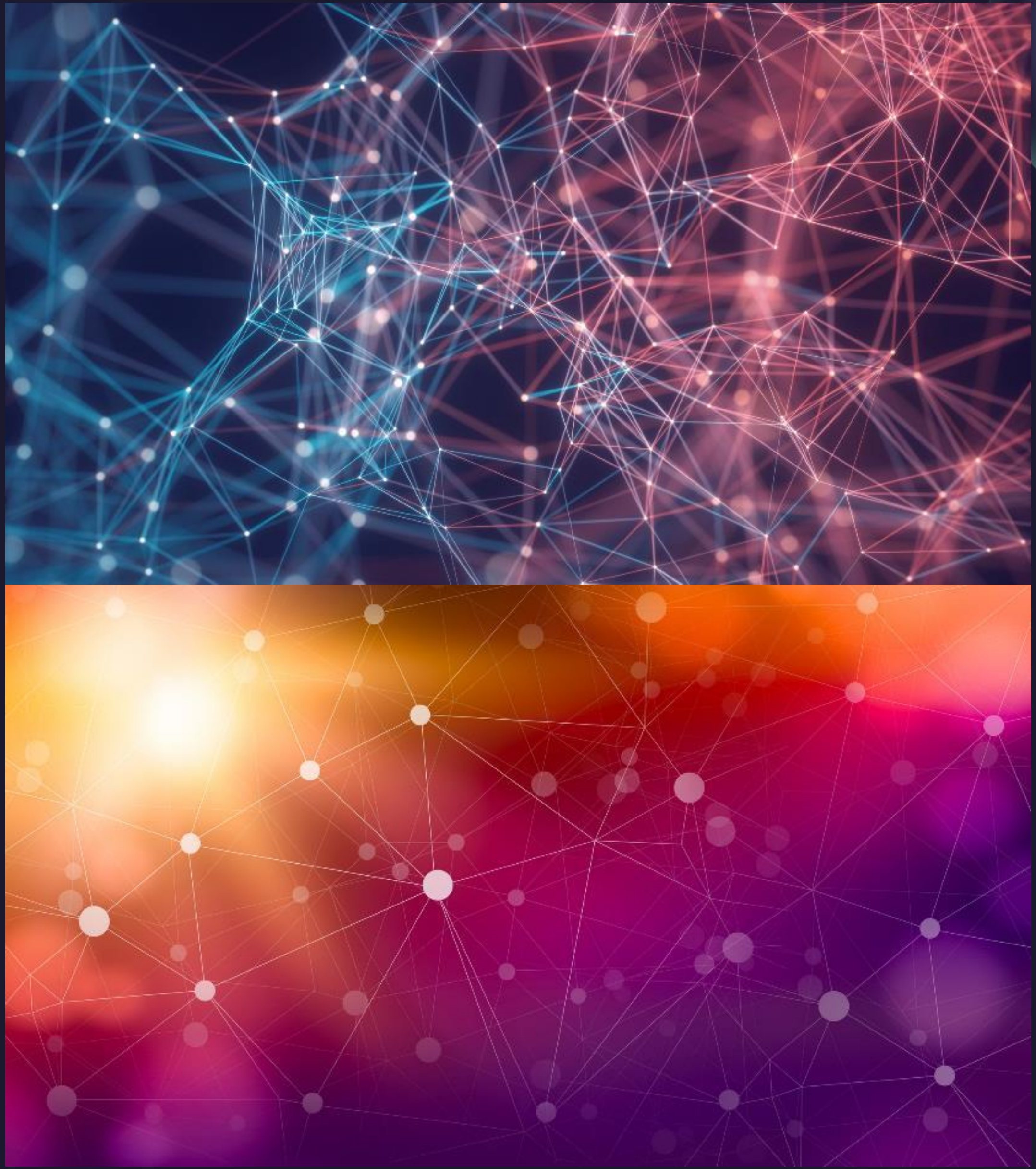
   POLAR CODES FOR ERROR CORRECTION, ANALYSIS AND DECODING ALGORITHMS.

2. Videos and Links:

- https://youtu.be/D_gadv-V-MQ
- https://youtu.be/Vj-1PwUNBek
- https://www.youtube.com/watch?v=S3bZOSlNFGo
- https://youtu.be/PNBFUV-ZetY
- https://github.com/cihatkececi/ChannelCodingProjectList#polar-codes

The End..
Thanks to Allah !

Any Questions?



Asmaa Gamal Abdel Halem Mabrouk Nagy
Communications & Electronics
15010473