CNU BLOCK

Based on the paper, When an LDPC decoder starts, the input is a set of **LLRs (log-likelihood ratios)** coming from the demodulator, which represent how reliable each received bit is. These LLRs are first given to the **Variable Node Units (VNUs)** as their starting values. The VNUs then process these values and generate **Variable-to-Check (VTC) messages**, which are passed along the edges of the Tanner graph to the **Check Node Units (CNUs)**. The CNUs receive these incoming messages and compute updated **Check-to-Variable (CTV) messages**, which are sent back to the VNUs. This exchange of messages between VNUs and CNUs continues for several iterations, gradually improving the estimates of the transmitted bits until all parity-check equations are satisfied and the final decoded bits are obtained.

CNU operation

The Check Node Unit (CNU) takes as input the Variable-to-Check (VTC) messages coming from the Variable Node Units. Each of these input messages contains two pieces of information: the sign, which indicates whether the bit is more likely a 0 or 1, and the magnitude, which shows how reliable that decision is.

From these inputs, the CNU generates Check-to-Variable (CTV) messages as outputs. Each output also has a sign, which is calculated as the overall parity (XOR) of all input signs except the one from its own edge, and a magnitude, which is selected using the minimum rule—if the input was the smallest value, the second smallest is used; otherwise, the smallest value is used.

Steps to perform the CNU operation in my code

1. First, the CNU determines the **overall sign** by calculating the parity (XOR) of the sign bits of all incoming **β messages** (VTC messages).
2. Next, it computes the **absolute values** of the β messages and identifies the **smallest (min1)** and the **second smallest (min2)** magnitudes.
3. Using these values, the CNU generates the outgoing **α messages** (CTV messages), which are its outputs.
4. For each outgoing α message, the CNU excludes the contribution of its own incoming β message:
    - if the excluded message corresponds to **min1**, the magnitude is set to **min2**.
    - otherwise, the magnitude is set to **min1**.
    - The final sign is obtained by multiplying the overall sign with the sign of the excluded β message.
5. The updated α messages are then stored in the **CTV memory** for use in the next decoding iteration.

Example:

Suppose CN0 Check node is connected to 4 variable nodes from VNU which beta messages.
Let CN0 is connected to VN0, VN1, VN2, VN3.
i.e VN0 = +4
VN1 = -2
VN2 = +3
VN3 = -5

Calculating overall sign parity.
First, we multiply sign of all bits i.e
i.e +4 = +
-2 = -
+3 = +
-5 = -
So, - is 1 and + is 0.
Xor operation 0^1^0^1 =0. i.e +

Now we look for the absolute value of 4 beta messages
i.e finding the smallest and second smallest magnitudes.
i.e min1 = 2 from VN1
min2 = 3 from VN2

Now computing for the alpha value
i.e $\alpha_{m,n} = T.\ min_{t \neq n} |\beta_i|$
where m = Index of the current check node unit
n = Inex of the variable node to which we are sending messages
β = message from VN_i to CN_m
α = message from CN_m to VN_n
T = Sign product

Now for each VN we exclude its own message and take the magnitude value of the other VN message add find the minimum value among them and multiply the minimum value with sign.
α of VN0 is,

Input	Magnitude	Minimum value (min)	α = sign * min
VN1 = -2	2	2	+2
VN2 = +3	3		
VN3 = -5	5		

α of VN1 is,

Input	Magnitude	Minimum value (min)	α = sign * min
VN0 = +4	4	3	+3
VN2 = +3	3		
VN3 = -5	5		

α of VN2 is,

Input	Magnitude	Minimum value (min)	α = sign * min
VN0 = +4	4	2	+2
VN1 = -2	2		
VN3 = -5	5		

α of VN3 is,

Input	Magnitude	Minimum value (min)	α = sign * min
VN0 = +4	4	2	+2
VN1 = -2	2		
VN2 = +3	3		

Finally, we have

VN0	+2
VN1	+3
VN2	+2
VN3	+2

These VN values are stored in CTV memory to compute for nest operation.