

Partitioning matrix and modifying min sum algorithm

Anurag Gupta
Microelectronics (2015-17)
Instructor: Prof Madhav P. Desai

IIT-Bombay
Department of Electrical Engineering



Min sum decoding algorithm



main.c

```
readMatrix() // reads the parity check matrix
readCodeBlock() // reads the input code block
minSumDecode() // decode the code block
findAccuracy() // check accuracy of decoded block
```



Tanner Graph

- LDPC code's parity check equations can be represented by bipartite graph, called the Tanner graph¹.

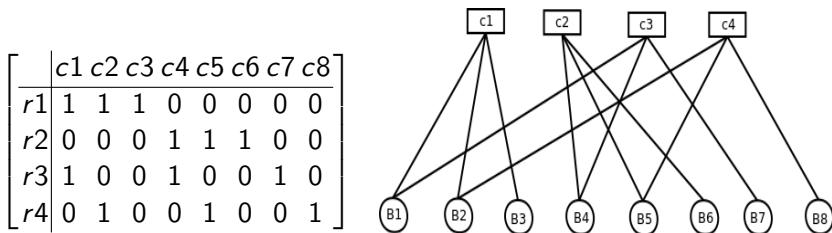


Figure: Tanner Graph

¹R. M. Tanner, "A recursive approach to low complexity codes," IEEE Trans. Inform. Theory, vol. IT-27, no. 5, pp. 533–547, September 1981.



A priori initialization

- A priories are calculated by soft information of the code bits.
- $aPriori[I] = -4 * C[I] * R * \frac{E_b}{N_o}$
- where $C[I] = i^{th}$ code block
- $R =$ code rate
- $\frac{E_b}{N_o} =$ signal to noise power ratio

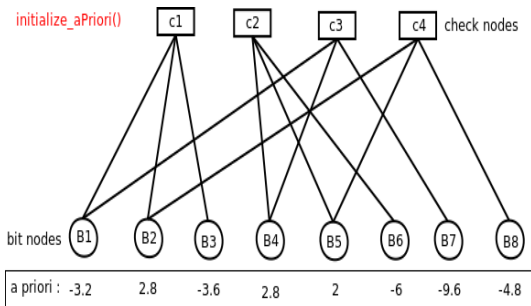


Figure: A priori initialization



Message initialization

- Messages are the information propagating from bit nodes to check nodes.
- These are initialized to a priori of their respective bit node.
- $message[I][J] = aPriori[I]$

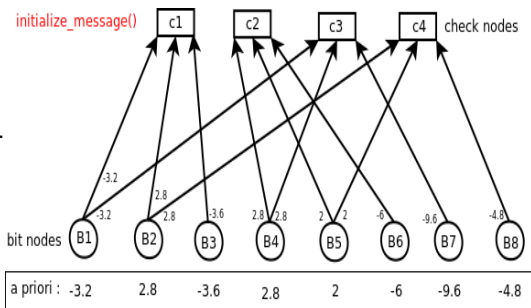
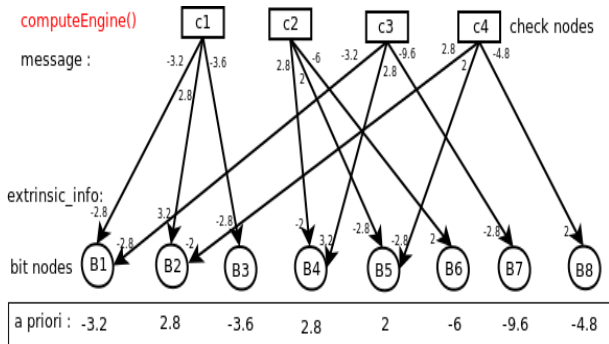


Figure: Message initialization



Extrinsic information calculation

- Extrinsic information of a bit node is calculated min sum of all the message's connected to that particular check node.

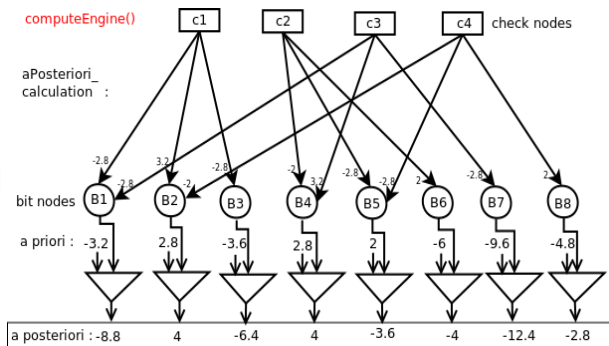


- $|E_{(j,i)}| = \text{Min}_{i' \in B_j, i' \neq i} |M_{j,i'}|$
- $\text{sign}(E_{(j,i)}) = \prod_{i' \in B_j, i' \neq i} \text{sign}(M_{j,i'})$



A posteriori calculation

- A posteriori probabilities are the output bit probabilities.
- These are used to modify the code block after every iteration.



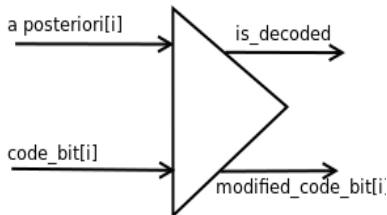
- $aPosteriori[I] = \sum_{j \in A_i} E_{j,i} + aPriori[I]$



isDecoded block

- This block flips a bit if it is different from hard decision of the a posteriori probability of the bit. Thus, modifies the code block.
- If, no bit got flipped then decoding stops.
- $is_decoded = 1$;
if $\forall i \text{ code_bit}[i] = \text{hard_decision}(\text{aPosteriori}[i])$

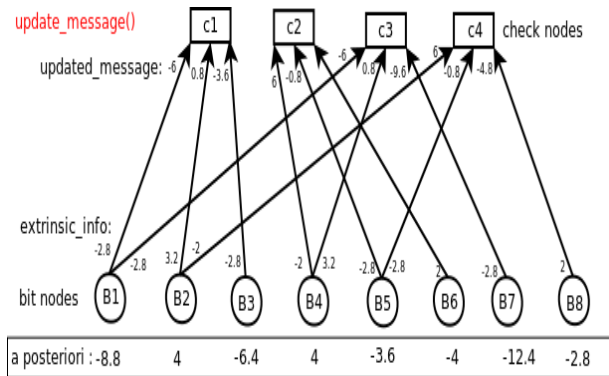
$is_decoded()$:





Updating messages

- Messages are updated and transmitted back to start the next iteration of decoding.



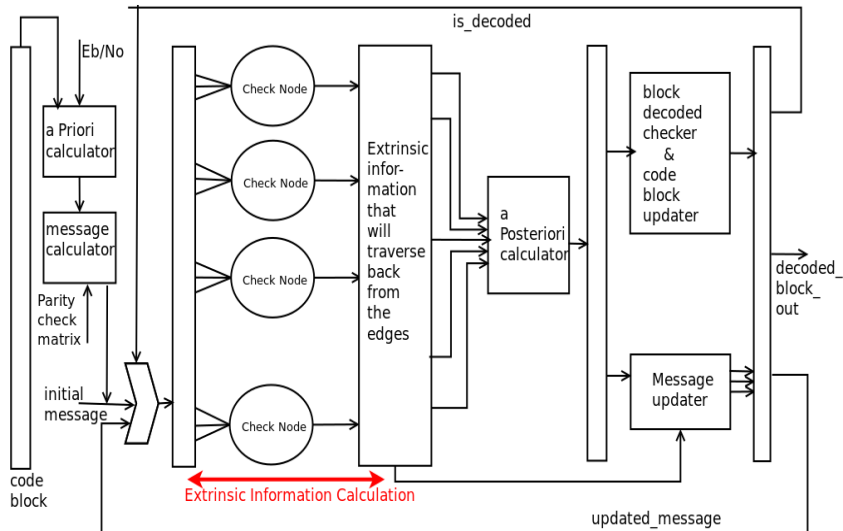
- $message_{(j,i)} = aPosteriori[i] - E_{(j,i)}$

minSumDecode() :

```

initialize_aPriori()
initializeMessage()
while  $nitr \geq Max\_nitr$  do
    initialize_aPosteriori()  $\Leftarrow aPriori$ 
    initializeExtrinsicInfo()  $\Leftarrow 0$ 
    checkNodeComputeEngine()
    is_decoded = checkIsDecode()
    if is_decoded = 1 then
        break
    else
        updateMessage()
    end if nitr ++
end while

```



Algorithmic complexity at each stage



- A priori calculation : $O(m)$
- Message calculation : $O(m \times p)$
- Extrinsic information calculation : $O(m \times p \times (p-1))$
 $\approx O(mp^2)$
- A posteriori calculation : $O(m \times p)$
- Message updation : $O(m \times p)$
- block decoded calculation & code block updation : $O(m)$



Partitioning matrix

- After partitioning the matrix we get four matrices as follows :

$$H = \left[\begin{array}{c|c} H_{11} & H_{12} \\ \hline H_{21} & H_{22} \end{array} \right]$$

- example :

$$\left[\begin{array}{c|cccccccc} & c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8 \\ \hline r1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ r2 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ r3 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ r4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \Rightarrow \left[\begin{array}{c|cccc|cccc} & c4 & c6 & c7 & c1 & c2 & c3 & c8 & c5 \\ \hline r2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ r3 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline r1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ r4 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array} \right]$$

- H12 and H21 are highly sparse.

Modified min sum algorithm



main.c

Require: Parity check matrix in row compressed form.

readMatrixH11()

readMatrixH12()

readMatrixH21()

readMatrixH22()

readCodeBlock1()

readCodeBlock2()

modifiedMinSumDecode()

findAccuracy()



A priori initialization

- A priories are calculated by soft information of the code bits.
- $aPriori[I] = -4 * C[I] * R * \frac{E_b}{N_o}$
- where $C[I] = i^{th}$ code block
- $R =$ code rate
- $\frac{E_b}{N_o} =$ signal to noise power ratio

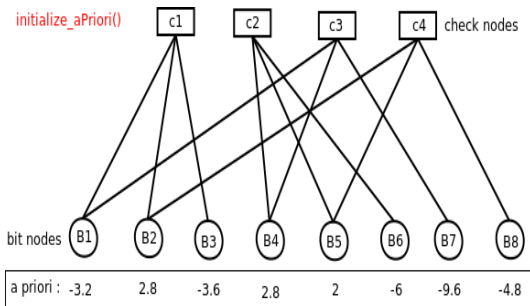
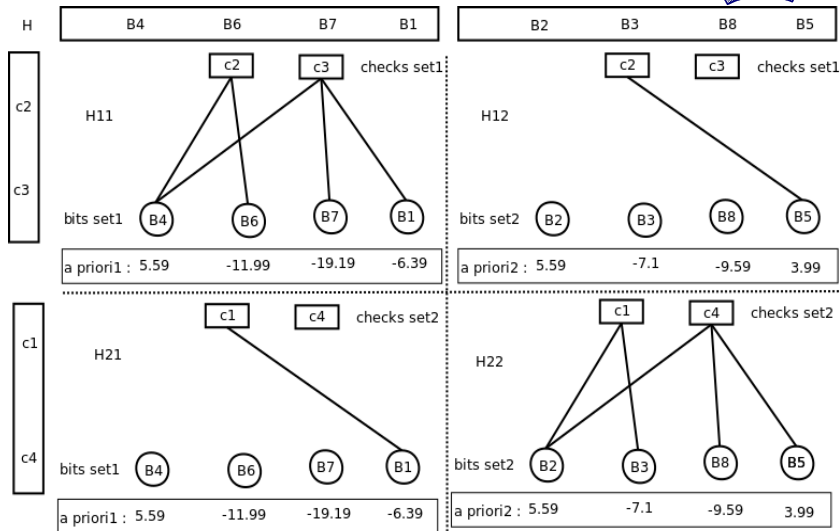


Figure: A priori initialization



A priori initialization





Message initialization

- Messages are the information propagating from bit nodes to check nodes.
- These are initialized to a priori of their respective bit node.
- $message[I][J] = aPriori[I]$

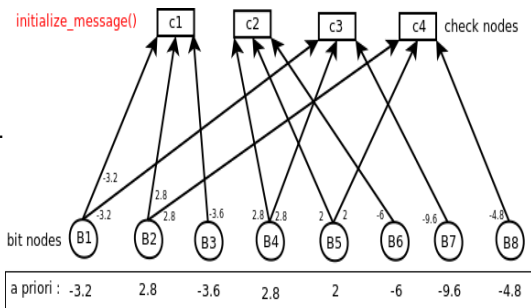
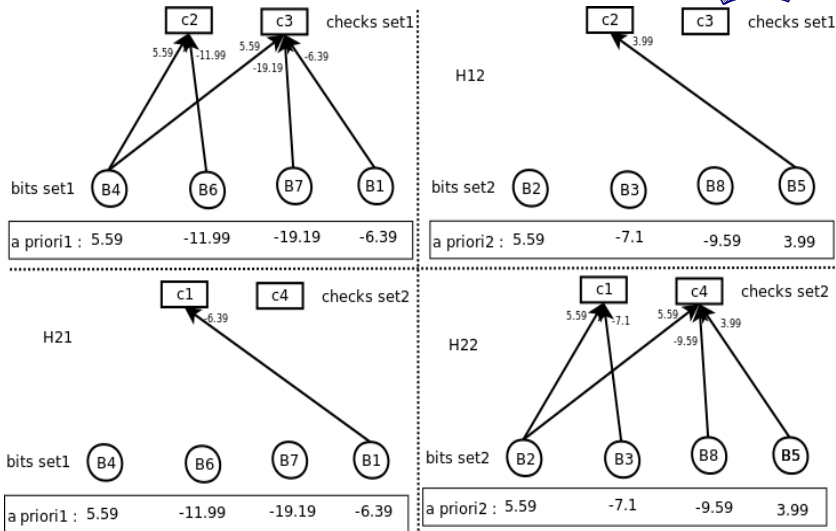


Figure: Message initialization



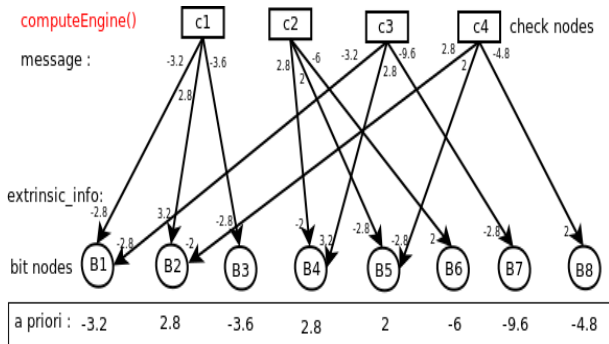
Message initialization





Extrinsic information calculation

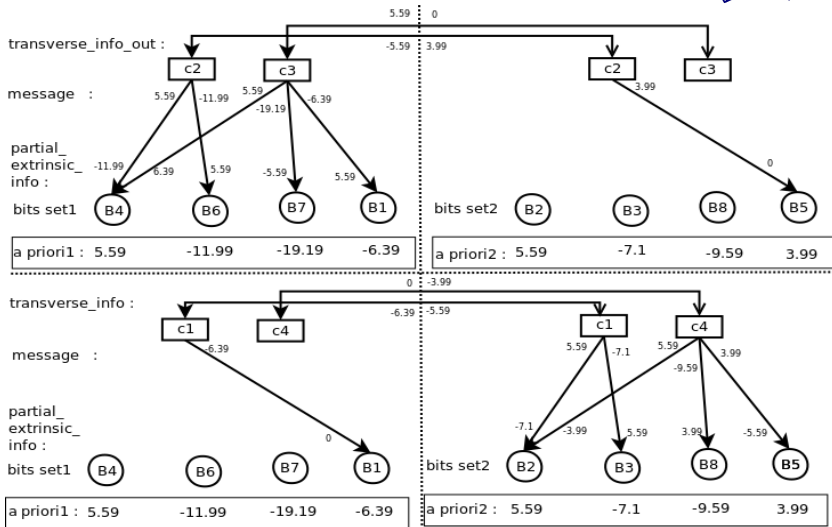
- Extrinsic information of a bit node is calculated min sum of all the message's connected to that particular check node.



- $|E_{(j,i)}| = \text{Min}_{i' \in B_j, i' \neq i} |M_{j,i'}|$
- $\text{sign}(E_{(j,i)}) = \prod_{i' \in B_j, i' \neq i} \text{sign}(M_{j,i'})$

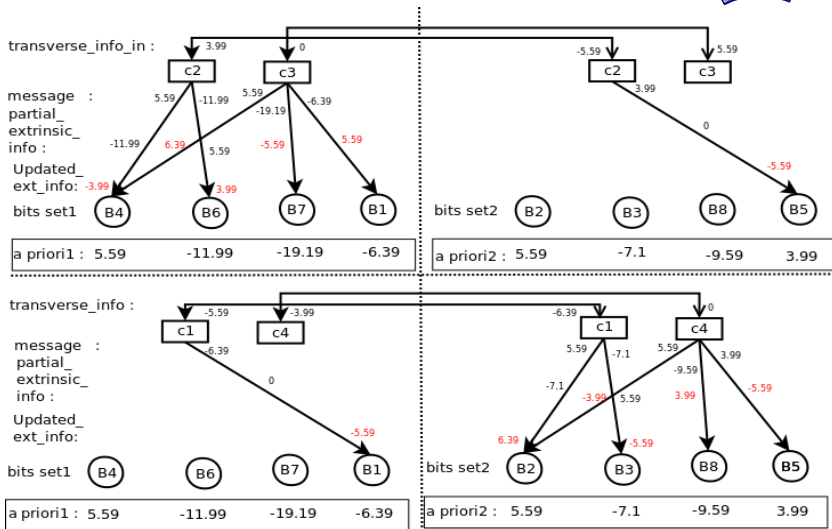


Partial Extrinsic information calculation





Update extrinsic information



modifiedMinSumDecode() :

```
initialize_aPriori(aPriori1)
initialize_aPriori(aPriori2)
initializeMessage(message11)
initializeMessage(message12)
initializeMessage(message21)
initializeMessage(message22)
while  $nitr \geq Max\_nitr$  do
    initialize_aPosteriori(aPosteriori1)  $\Leftarrow$  aPriori1
    initialize_aPosteriori(aPosteriori2)  $\Leftarrow$  aPriori2
    initializeExtrinsicInfo(ext_info11)  $\Leftarrow$  0
    initializeExtrinsicInfo(ext_info12)  $\Leftarrow$  0
    initializeExtrinsicInfo(ext_info21)  $\Leftarrow$  0
    initializeExtrinsicInfo(ext_info22)  $\Leftarrow$  0
    ...
end while
```

modifiedMinSumDecode() :

while ... do

...

computeEngine(H11, message11, ext_info11, trans_info11_12)

computeEngine(H22, message22, ext_info22, trans_info22_12)

computeEngine(H12, message12, ext_info12, trans_info12_11)

computeEngine(H21, message21, ext_info21, trans_info21_22)

transverseCorrection(H11, transverse_info12_11, ext_info11)

transverseCorrection(H22, transverse_info21_22, ext_info22)

transverseCorrection(H21, transverse_info22_21, ext_info21)

transverseCorrection(H12, transverse_info11_12, ext_info12)

update_aPosteriori(H11, ext_info11, aPosteriori1)

update_aPosteriori(H22, ext_info22, aPosteriori2)

update_aPosteriori(H12, ext_info12, aPosteriori1)

update_aPosteriori(H21, ext_info21, aPosteriori2)

...

modifiedMinSumDecode() :

while ... do

...

is_decoded1 = checkIsdecoded(code_block1, aPosteriori1)

is_decoded2 = checkIsdecoded(code_block2, aPosteriori2)

if (*is_decoded1*&&*is_decoded2*) == 1 **then**

break

else

updateMessage(ext_info11, aPosteriori1, message11)

updateMessage(ext_info22, aPosteriori2, message22)

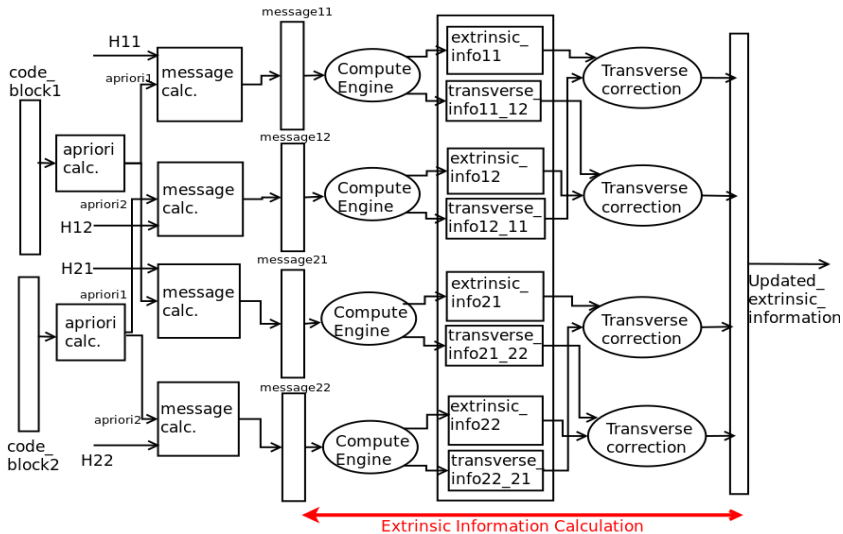
updateMessage(ext_info12, aPosteriori1, message12)

updateMessage(ext_info21, aPosteriori2, message21)

end if

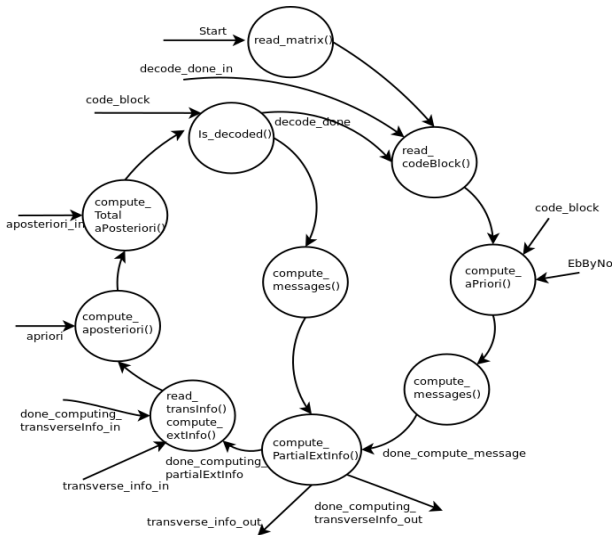
nitr ++

end while





State Diagram



Matrix Initialization



- Maximum file length till matrix order $12k$ is $\approx 60k$, thus we have to store $60k$ values of type `uint16_t`.
- thus taking 4 memory of $20k$ each .
`uint16_t mem[20,000]`
- populate it with reading a pipe of size 64bits.
- Thus writing 4 word at a time.
- $\approx (60k/4)/4$ clocks to write matrix in memory.

Code block input



- Code block file length till matrix order 12k is $\approx 13k$, thus we have to store $\approx 13k$ values of bit data type.
- We can choose a thick pipe width of 256 bits or 512 bits. Thus it will take upto 64 or 32 clock cycles respectively to input a code block.