Partitioning matrix and modifying min sum algorithm

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

IIT-Bombay
Department of Electrical Engineering

June 9, 2017

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 graph[1].

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

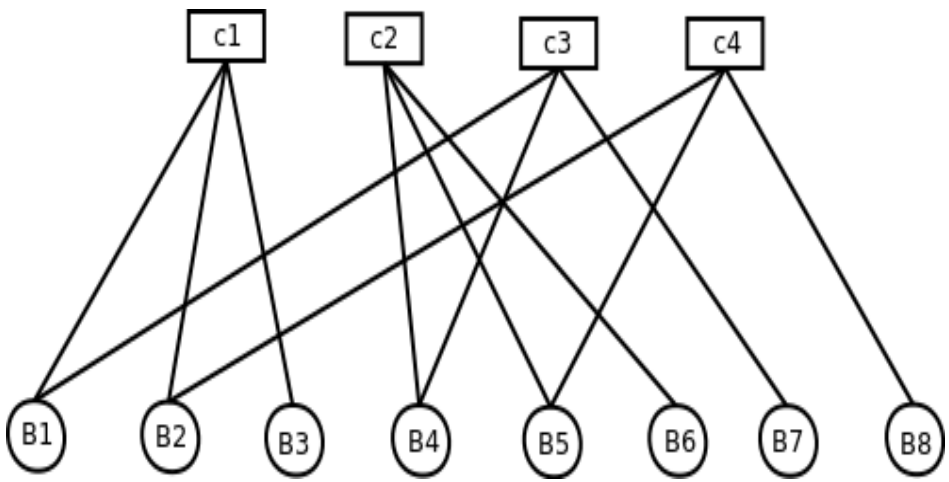


Figure: Tanner Graph

[1] R. 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.
- $aPriori[I] = -4 * C[I] * R * \frac{Eb}{No}$
- where C[I] = i^{th} code block
- R = code rate
- $\frac{Eb}{No}$ = signal to noise power ratio

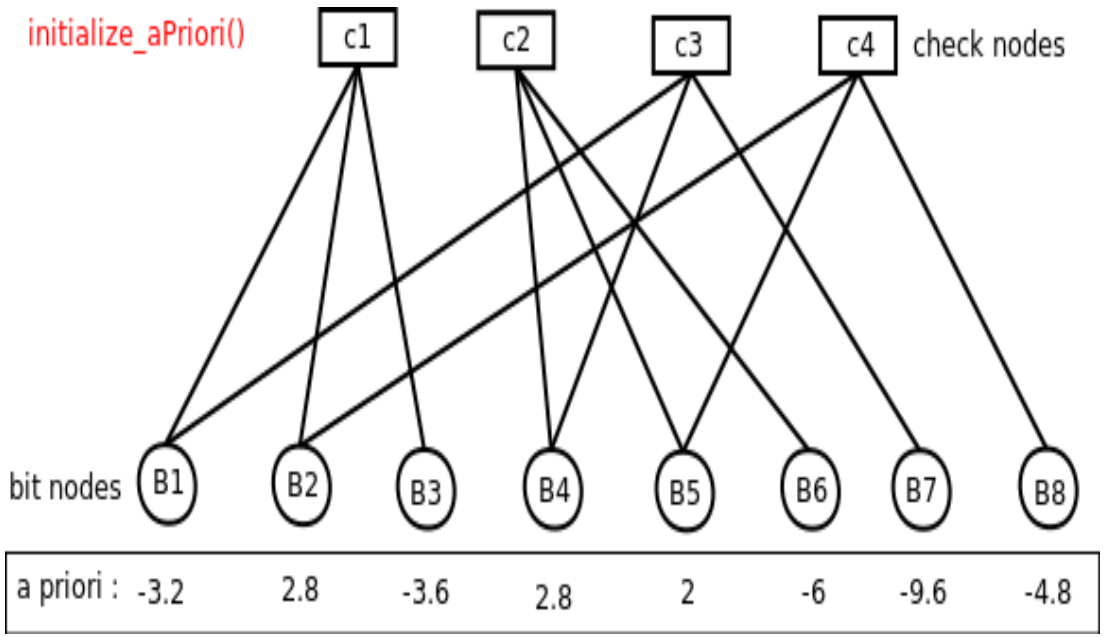


Figure: A priori initialization

Message initialization

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

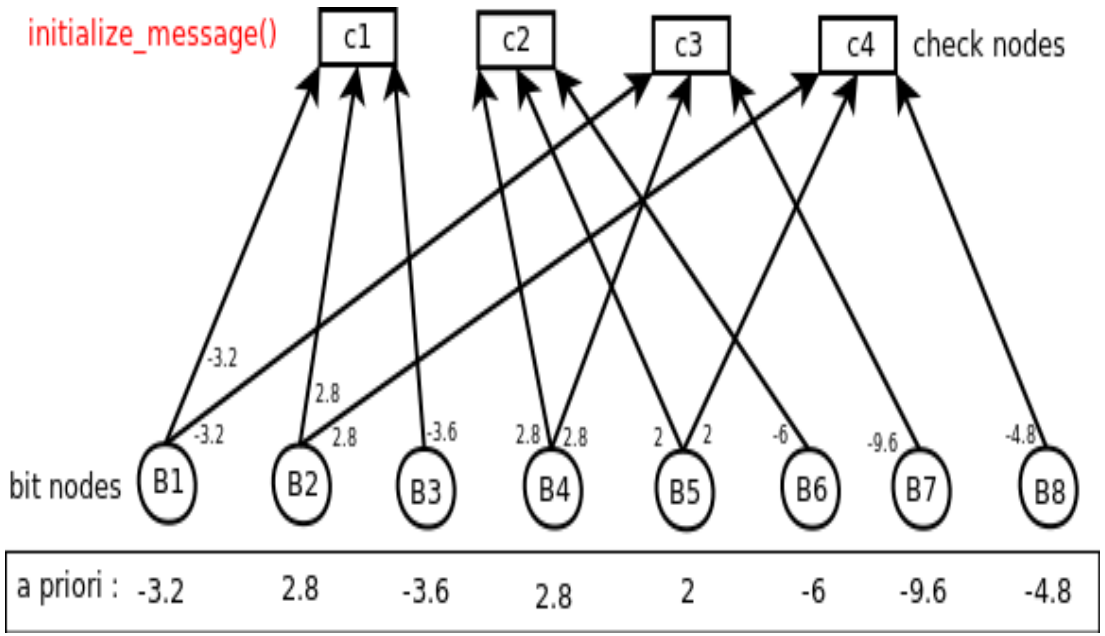
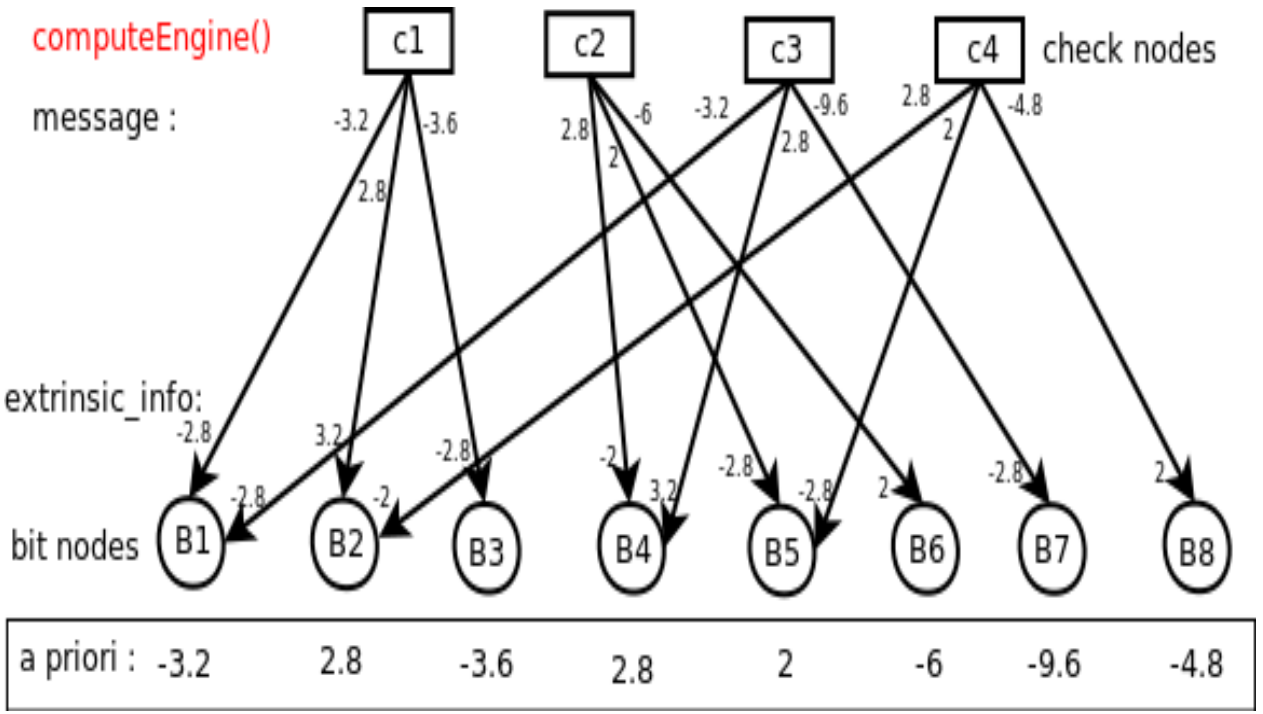


Figure: Message initialization

Extrinsic information calculation

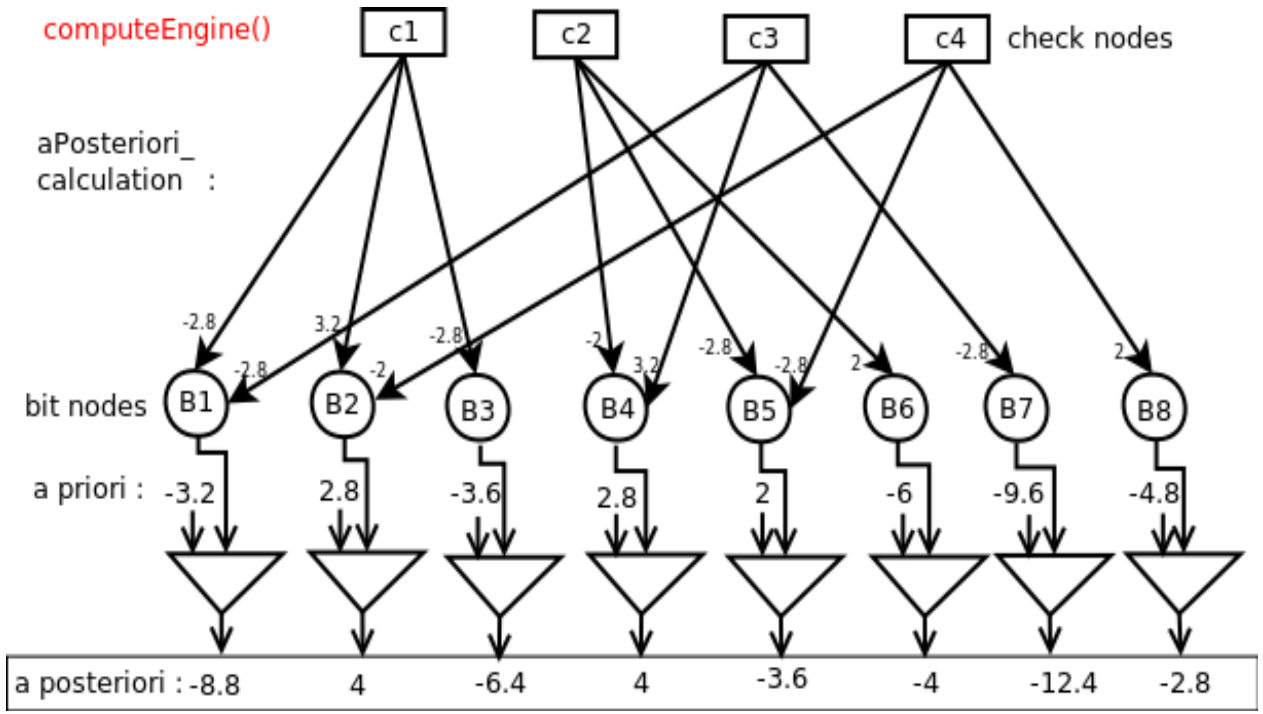
- Extrinsic information of a bit node is calculated min sum of all the message's connected to that particular check node.



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

A posteriori calculation

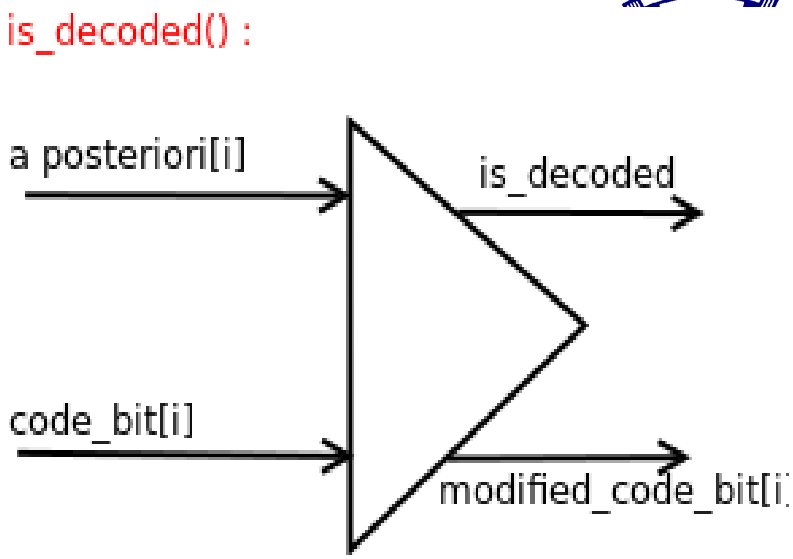
- A posteriori probabilities are the output bit probabilities.
- These are used to modify the code block after every iteration.



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

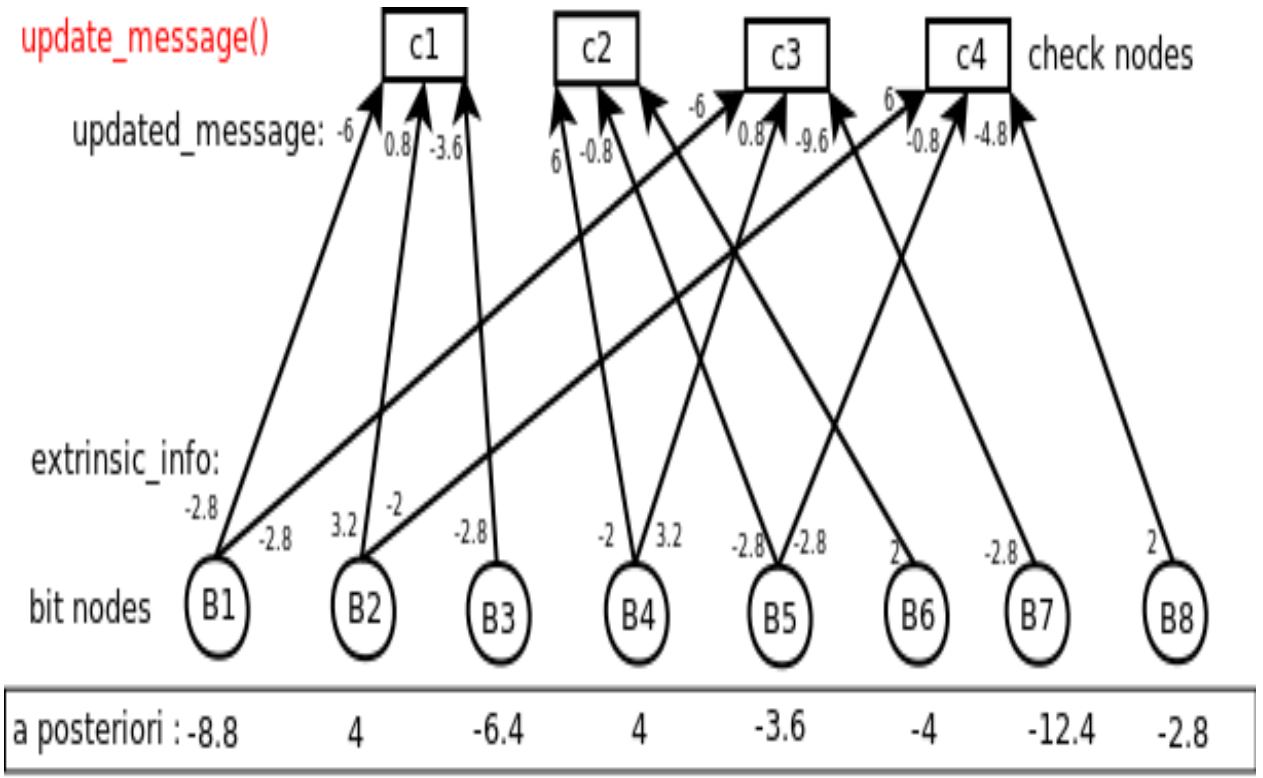
isDecoded block

- This block flips a bit if it is different form 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 ∀ i code_bit[I] = hard_decision(aPosteriori[I])



Updating messages

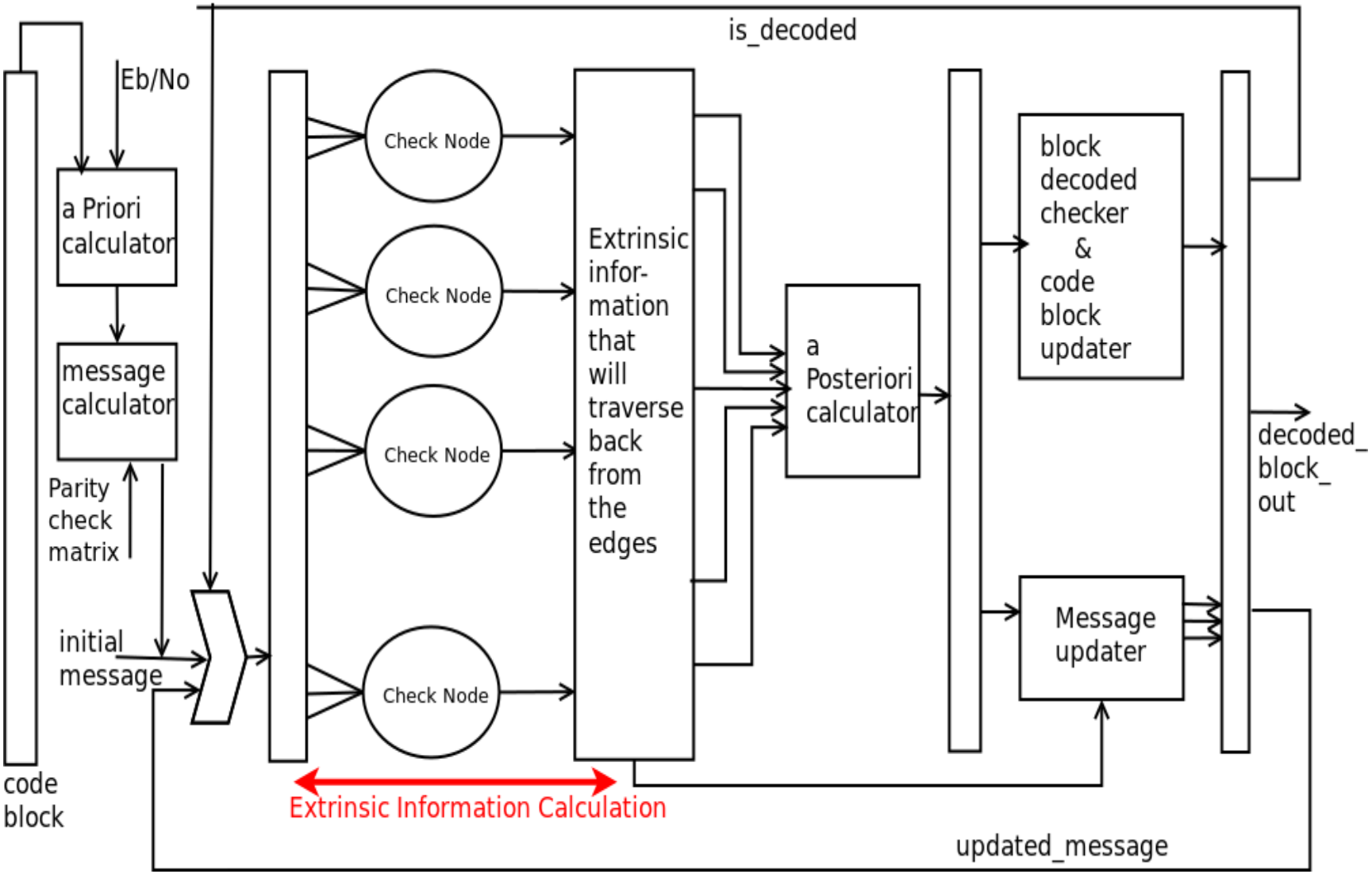
- Messages are updated and transmitted back to start the next iteration of decoding.



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

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


is_decoded
Eb/No
a Priori calculator
message calculator
Parity check matrix
initial message
code block
Check Node
Check Node
Check Node
Check Node
Extrinsic infor-mation that will traverse back from the edges
a Posteriori calculator
block decoded checker & code block updater
Message updater
decoded_ block_ out
Extrinsic Information Calculation
updated_message

Algorithmic complexity at each stage

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

Partitioning matrix

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

$$H = \left[\begin{array}{c|c} H11 & H12 \\ \hline H21 & H22 \end{array}\right]$$

- 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.
- $aPriori[I] = -4 * C[I] * R * \frac{Eb}{No}$
- where C[I] = i^{th} code block
- R = code rate
- $\frac{Eb}{No}$ = signal to noise power ratio

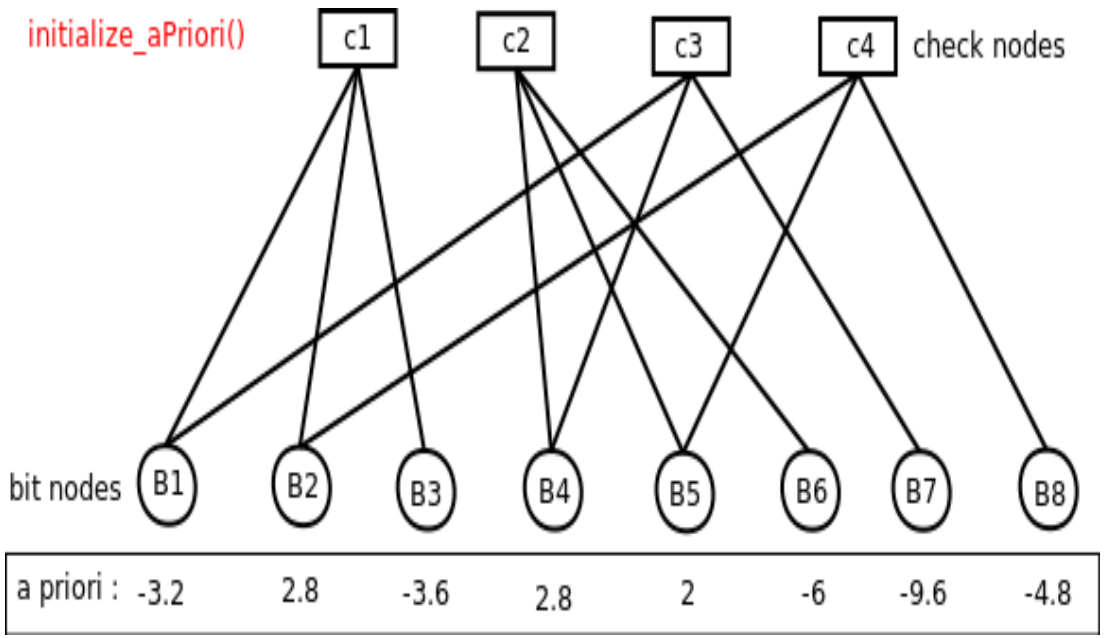
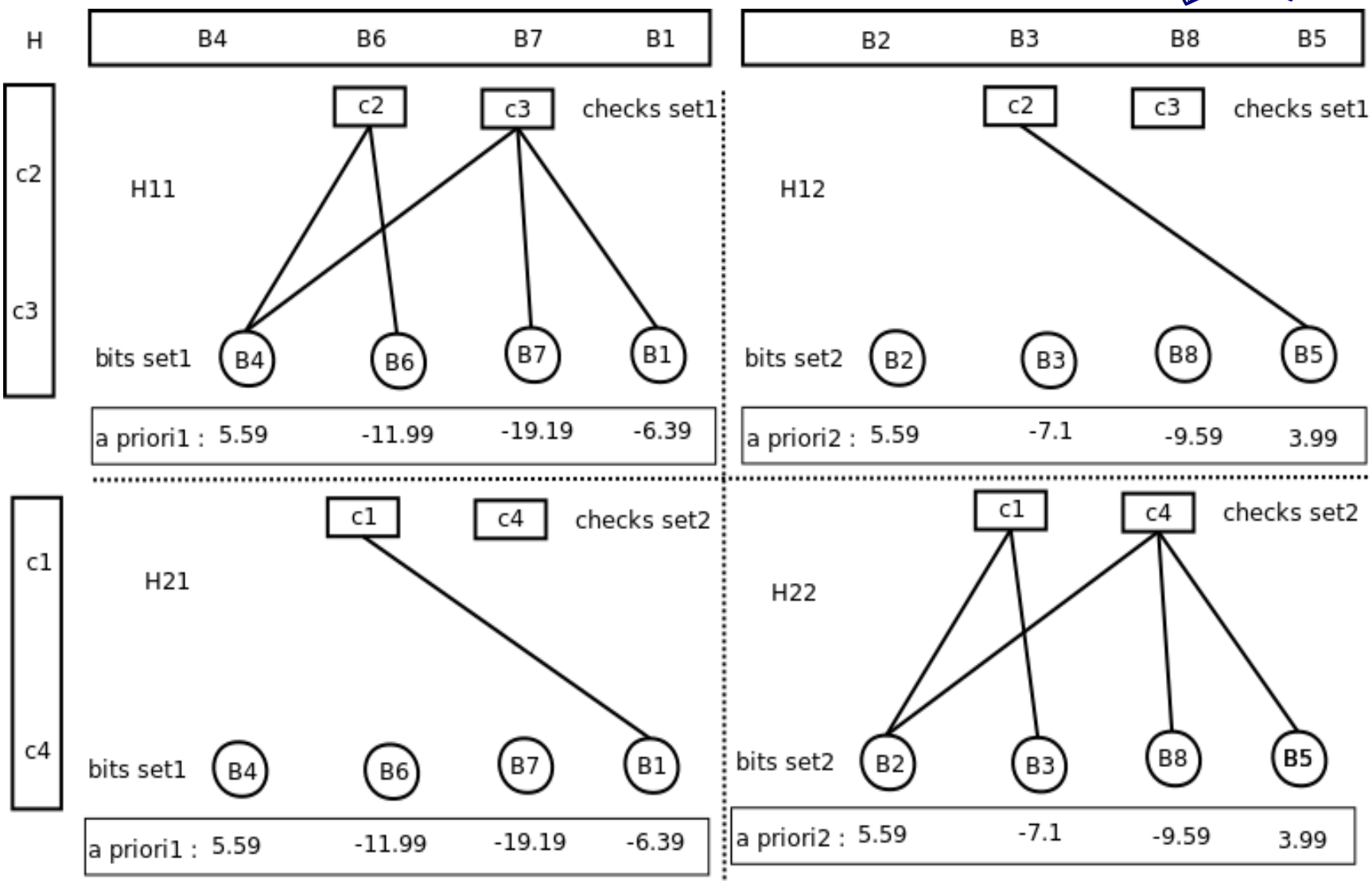


Figure: A priori initialization

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.
- $message[I][J] = aPriori[I]$

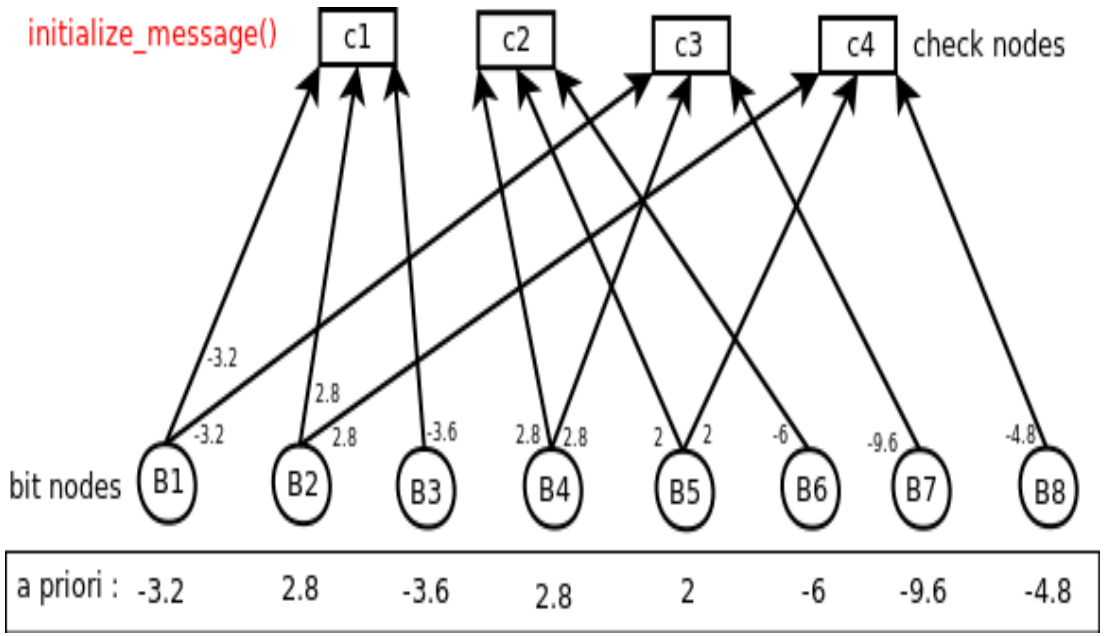
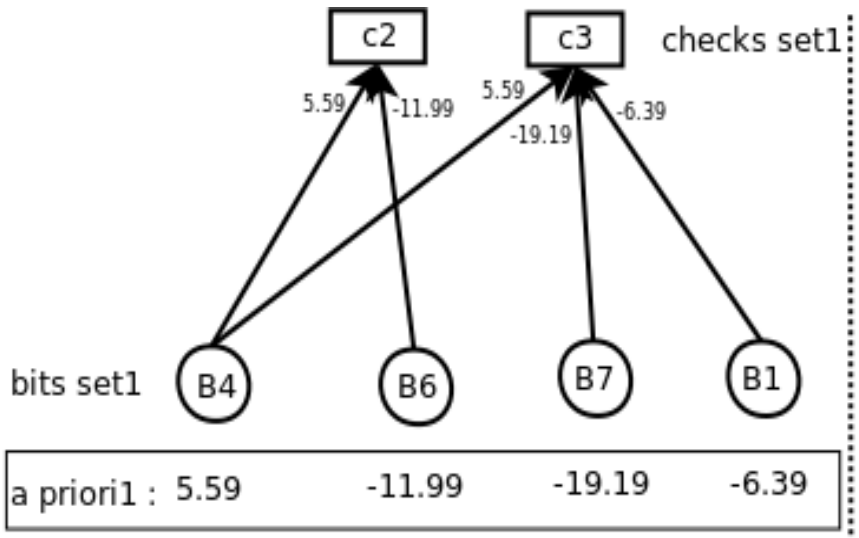
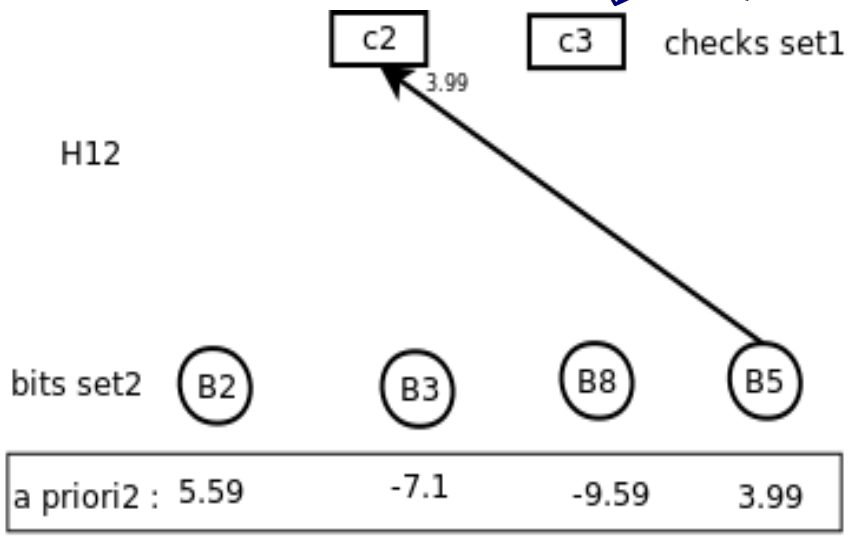
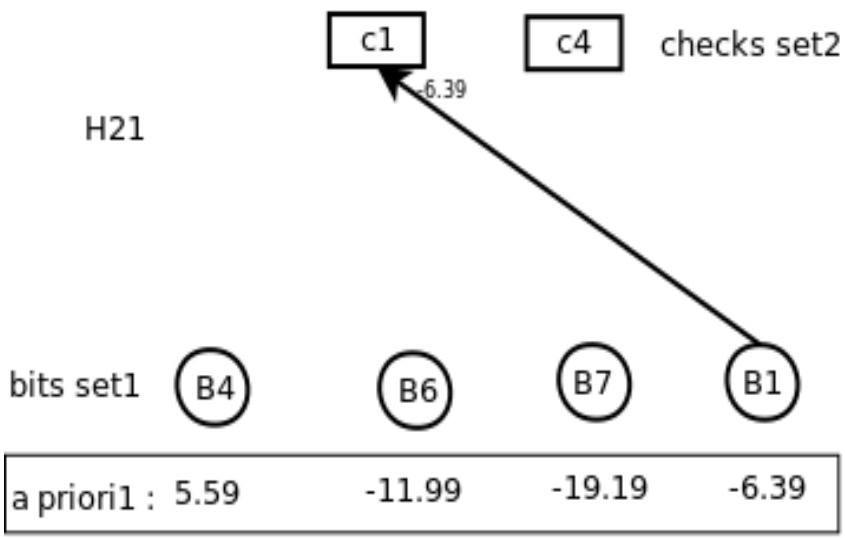


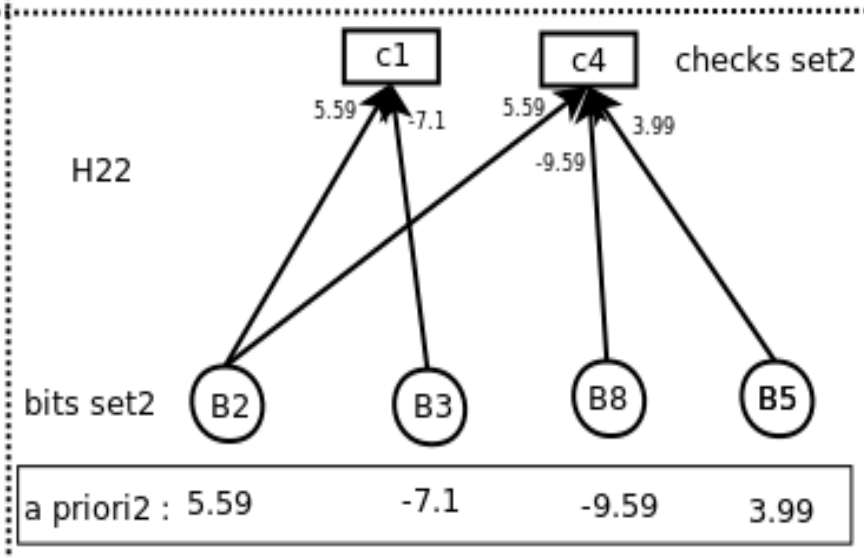
Figure: Message initialization

Message initialization



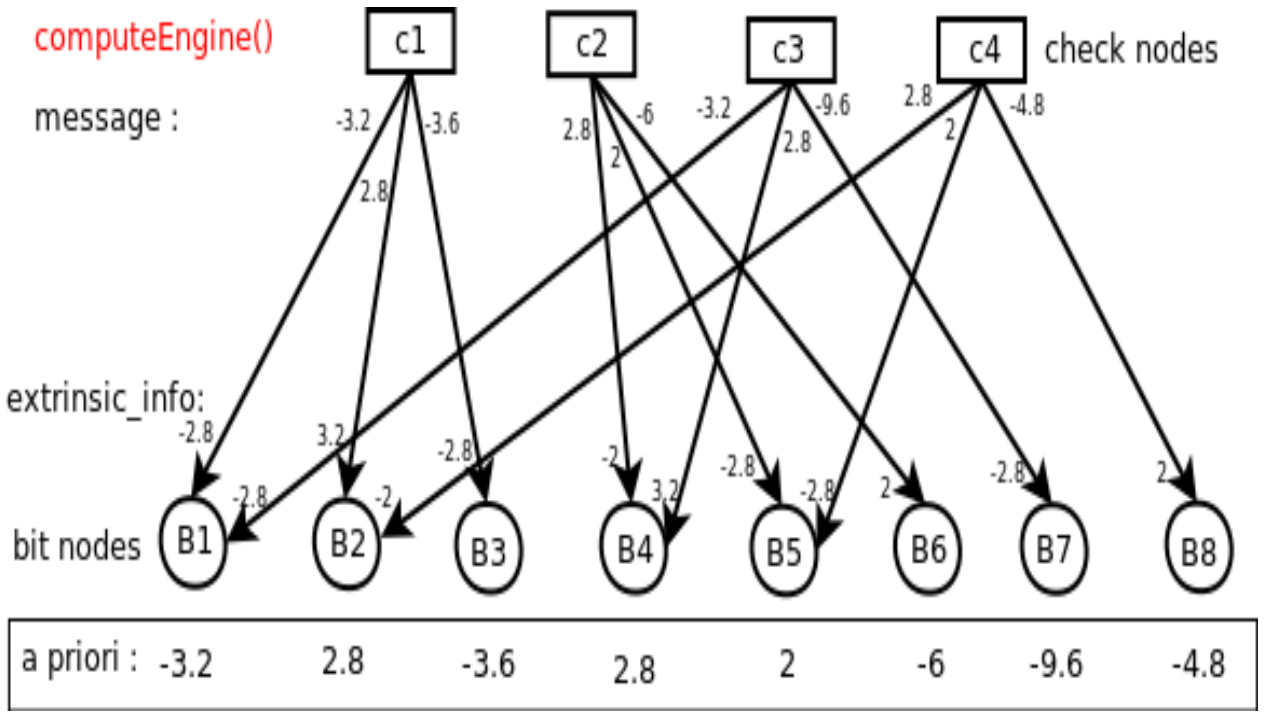






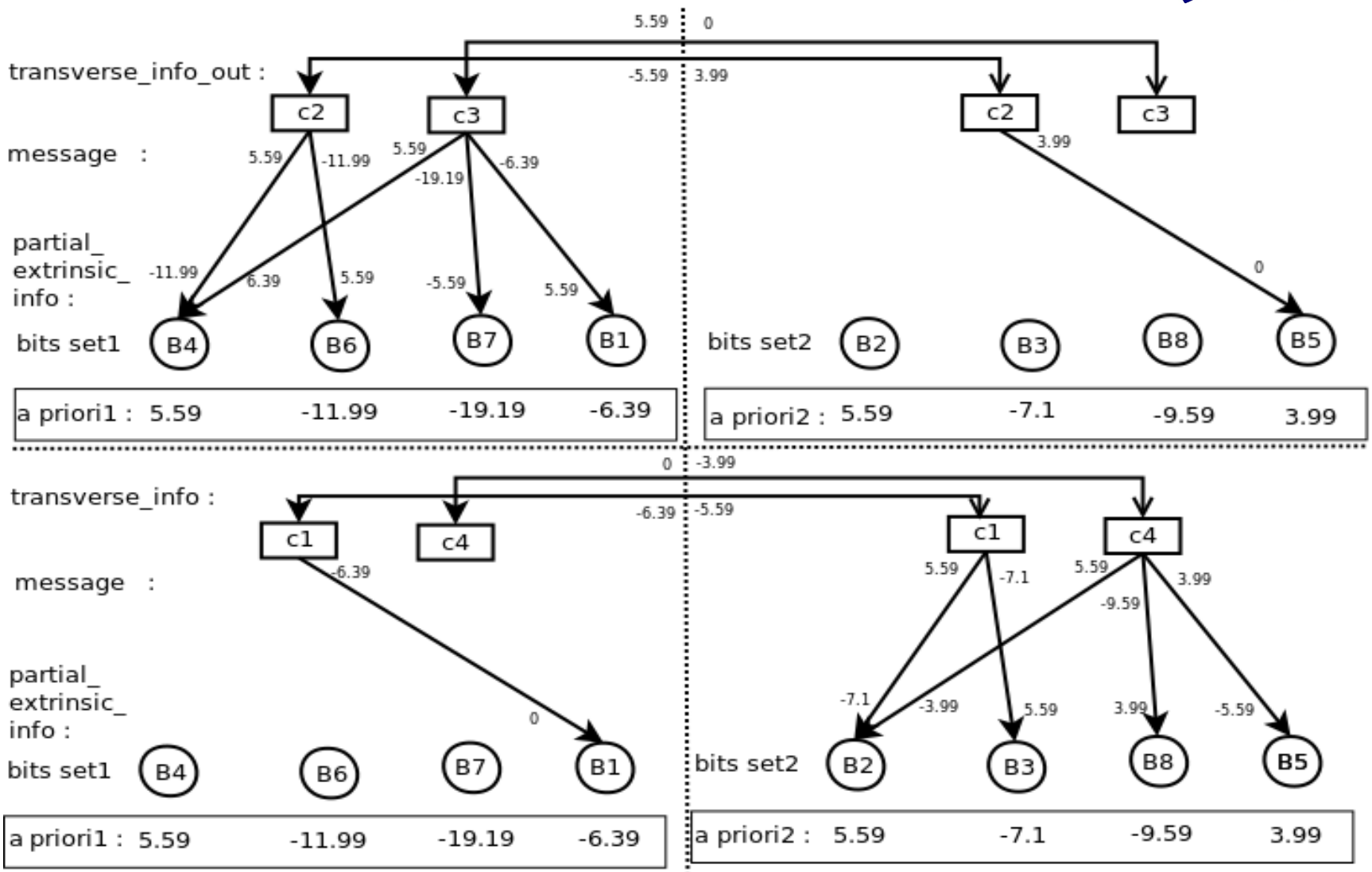
Extrinsic information calculation

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

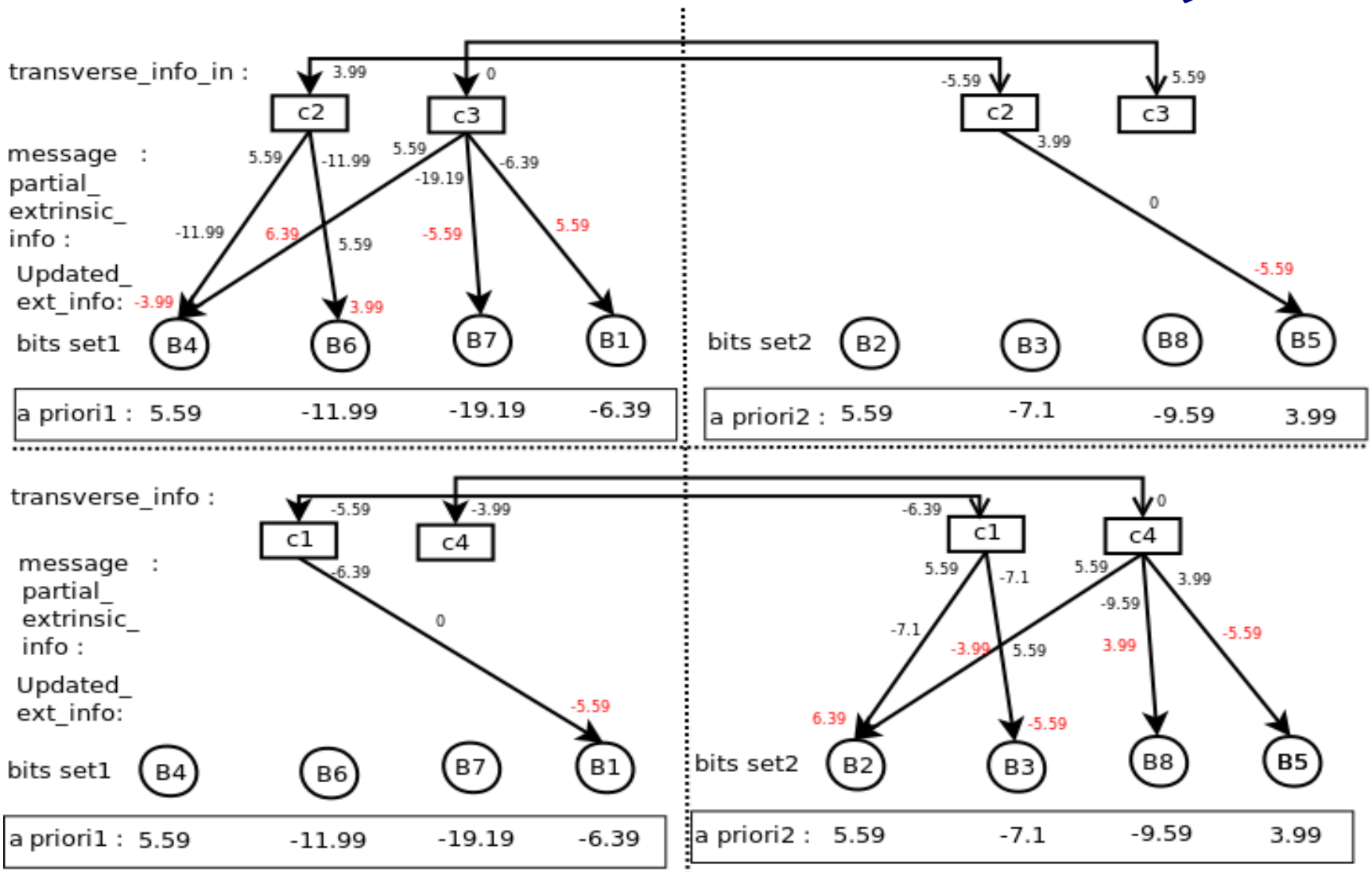


- $|E_{(j,i)}| = Min_{i' \in B_j \ i' \neq i} |M_{j,i'}|$
- $sign(E_{(j,i)}) = \prod_{i' \in B_j \ i' \neq i} sign(M_{j,i'})$

Partial Extrinsic information calculatio



Update extrinsic information



modifiedMinSumDecode() :

initialize_aPriori(*aPriori*1)
initialize_aPriori(*aPriori*2)
initializeMessage(*message*11)
initializeMessage(*message*12)
initializeMessage(*message*21)
initializeMessage(*message*22)
while $nitr \geq Max\_nitr$ **do**
  initialize_aPosteriori(*aPosteriori*1) $\Leftarrow$ *aPriori*1
  initialize_aPosteriori(*aPosteriori*2) $\Leftarrow$ *aPriori*2
  initializeExtrinsicInfo(*ext_info*11) $\Leftarrow 0$
  initializeExtrinsicInfo(*ext_info*12) $\Leftarrow 0$
  initializeExtrinsicInfo(*ext_info*21) $\Leftarrow 0$
  initializeExtrinsicInfo(*ext_info*22) $\Leftarrow 0$
  ...
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

$updateMessage(ext\_info11, aPosteriori1, message11)$

$updateMessage(ext\_info22, aPosteriori2, message22)$

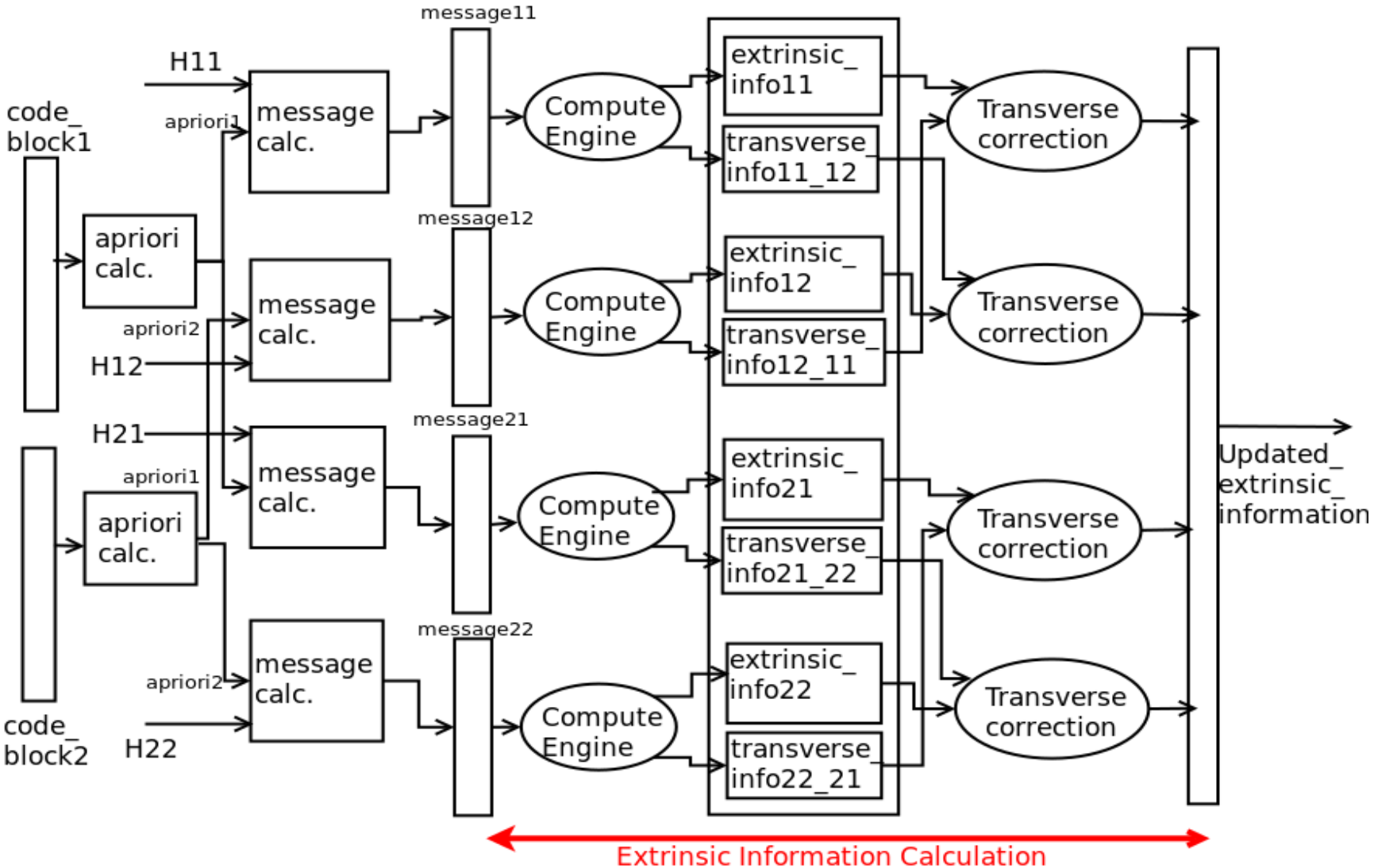
$updateMessage(ext\_info12, aPosteriori1, message12)$

$updateMessage(ext\_info21, aPosteriori2, message21)$

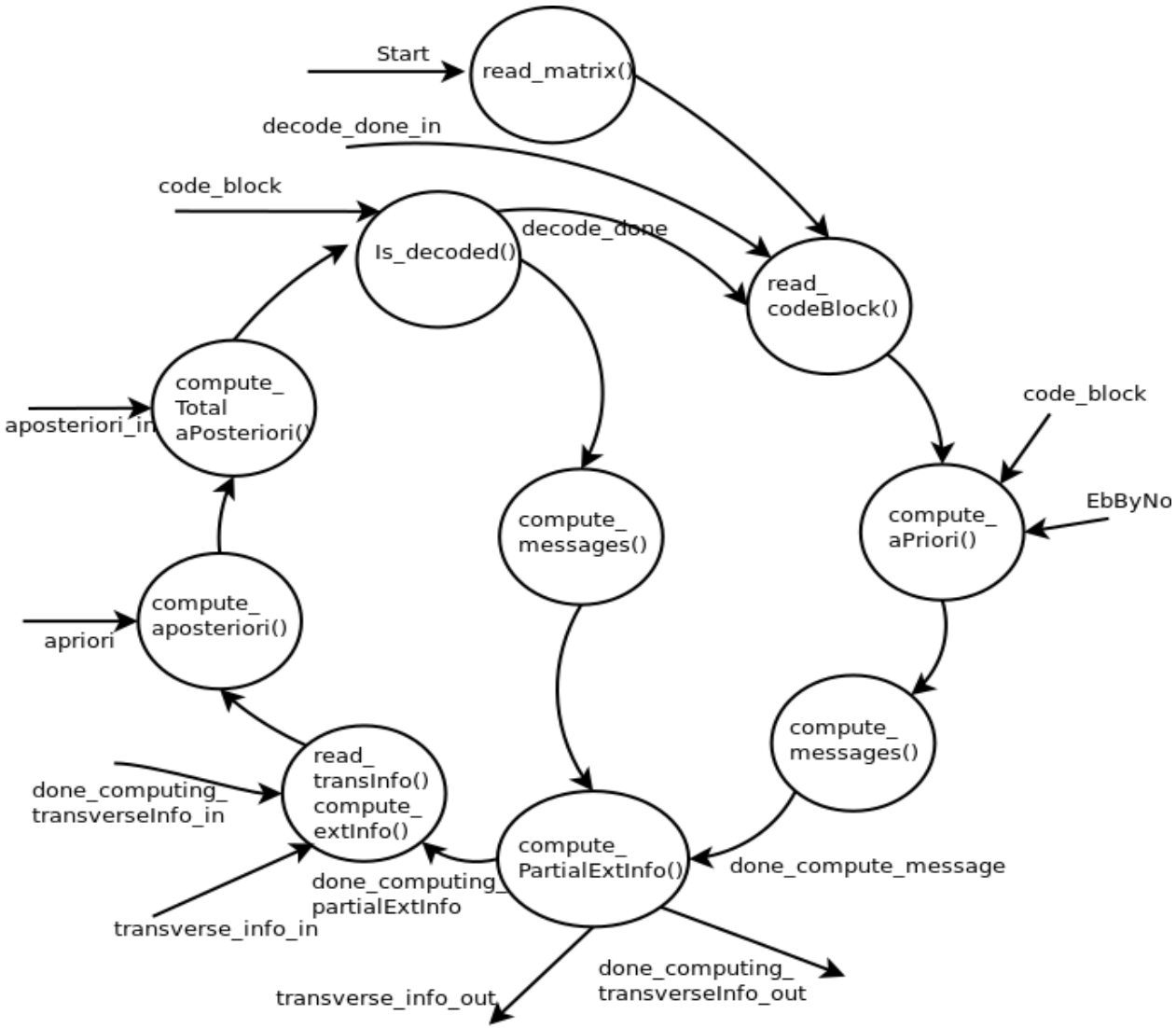
end if

$nitr++$

end while


code_
block1
code_
block2
apriori
calc.
apriori
calc.
H11
apriori1
H12
apriori2
H21
apriori1
H22
apriori2
message
calc.
message
calc.
message
calc.
message
calc.
message11
message12
message21
message22
Compute
Engine
Compute
Engine
Compute
Engine
Compute
Engine
extrinsic_
info11
transverse_
info11_12
extrinsic_
info12
transverse_
info12_11
extrinsic_
info21
transverse_
info21_22
extrinsic_
info22
transverse_
info22_21
Transverse
correction
Transverse
correction
Transverse
correction
Transverse
correction
Updated_
extrinsic_
information
Extrinsic Information Calculation

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 ≈13k, thus we have to store ≈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.