Implementation of LDPC Decoder - Using AHIR Tool Chain

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

IIT-Bombay
Department of Electrical Engineering

June 30, 2017

Outline

Motivation

Why LDPC ?

- Channel capacity approaching codes[1]
  - S.Y.Chung shannon limit approaching code: For a white Gaussian noise channel threshold within 0.0045 dB of the Shannon limit with block length of 10^7.
- Decoding time varies linearly proportional to block length
  - Parity check matrix is sparse
- Decoder can be parallelized
  - Decoding algorithms are iterative

[1] S.-Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. L. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," IEEE Commun. Letters, vol. 5, no. 2, pp. 58–60, February 2001.

LDPC Decoding Algorithms

- Hard decoding algorithms
  - Bit flipping algorithm
- Soft decoding algorithms
  - Sum product decoding
  - Min sum decoding

Min Sum Decoding Algorithm

[2] R. M. Tanner, "A recursive approach to low complexity codes," IEEE Trans. Inform. Theory, vol. IT-27, no. 5,

Min Sum Decoding Algorithm

Tanner Graph:

- LDPC code's parity check equations can be represented by bipartite graph, called the Tanner graph[2].

$$
\begin{array}{c}
\begin{array}{cccccccc} c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8 \end{array} \\
\begin{array}{c} r1 \\ r2 \\ r3 \\ r4 \end{array}
\begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}
\end{array}
$$

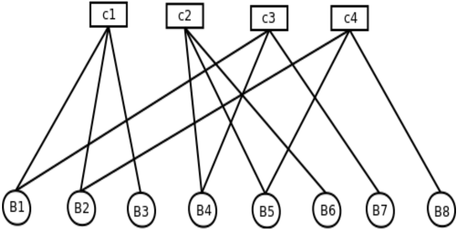


Figure: Tanner Graph

[2] R. M. Tanner, "A recursive approach to low complexity codes," IEEE Trans. Inform. Theory, vol. IT-27, no. 5,

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

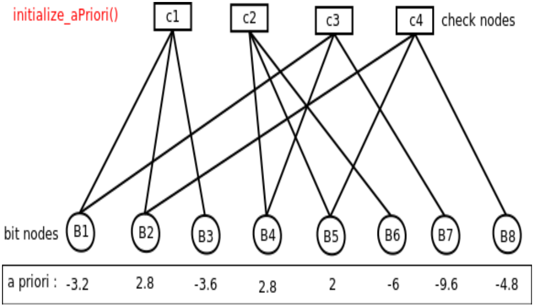


Figure: A priori initialization

Message initialization

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

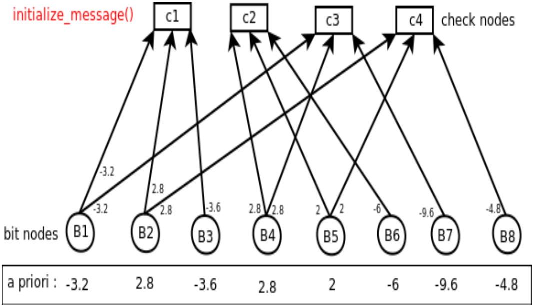
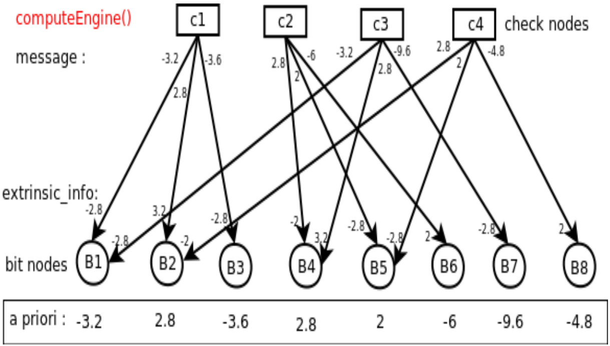


Figure: Message initialization

Extrinsic information calculation

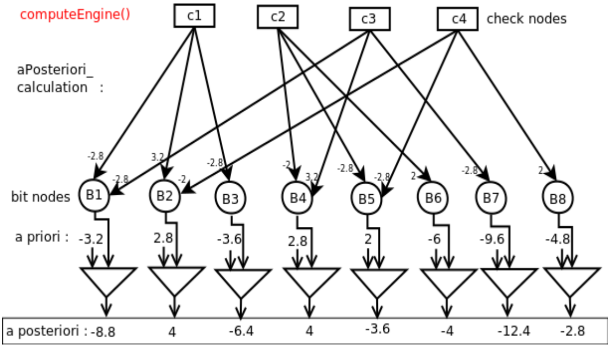
- Extrinsic information of a bit node is calculated as min sum of all the messages connected to that particular check node.



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

A posteriori calculation

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

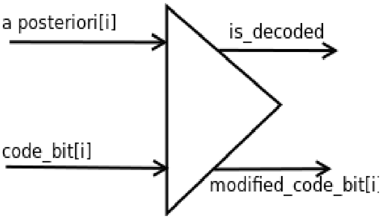


- $aPosteriori[I] = \sum_{j \in A_i} E_{j,i} + aPriori[I]$

isDecoded block

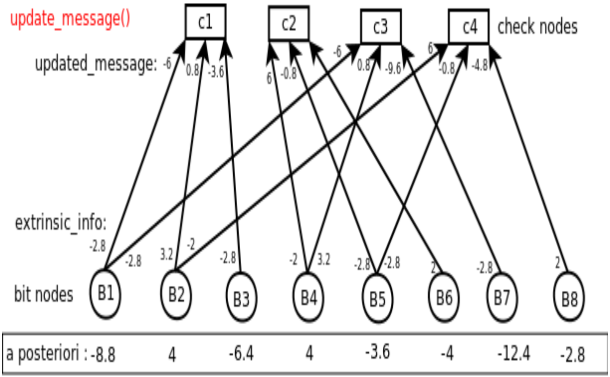
- This block flips a bit if it is different form hard decision of the a posteriori probability of the bit. Thus, modifies the code block.
- If, no bit got flipped then decoding stops.
- $is\_decoded = 1$;
  if $\forall$ i code_bit[I] = hard_decision(aPosteriori[I])

is_decoded() :



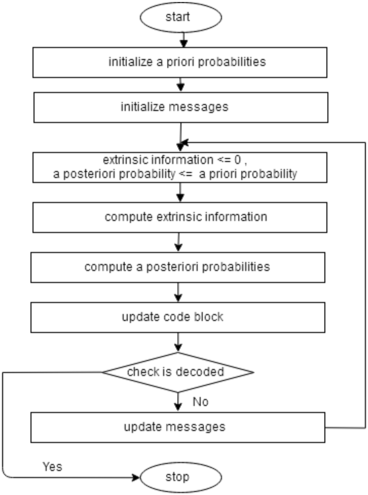
Updating messages

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



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

LDPC Decoding : Min Sum Decode



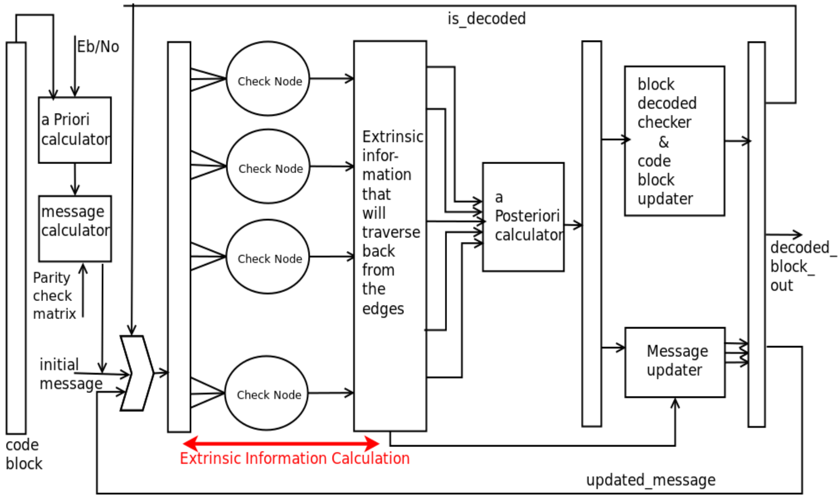
Decoder Implementation

Three different implementation[3] strategies are possible.

1. Serial decoder
   - simple
   - cheap
   - slow
2. Fully parallel decoder
   - complex
   - costly
   - Super fast
3. Partial parallel decoder.
   "Can we effectively partition a bipartite graph corresponding to a LDPC parity check matrix ?"

[3] Muhammad Awais and Carlo Condo, "Flexible LDPC Decoder Architectures," VLSI Design, vol. 2012, Article ID 730835, 16 pages, 2012.

Implementation of Serial Decoder


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

C Level Implementation & Testing

Serial Min Sum Decoder - Quasi-Cyclic Matrix

- Min sum algorithm is implemented for Gaussian channel.
- Quasi-cyclic matrix of block size(n=) 4K, 8K and 12K are formed using Sridhara-Fuja-Tanner algorithm.
- Five different code rates(R=) 0.75, 0.80, 0.85, 0.90 and 0.95 are taken.
- Raw input bit error rate(BER(IN)) is between 10^{-2} to 10^{-3}, converted in form of Eb/No(db) to express input SNR in db.
- BER(OUT) : Output block error rate.
- We have tabulated when the first code block get wrongly decoded till 1 million transmitted blocks.

First error till 1 million blocks

n≃	BER(In)≃	R=0.75	R=0.80
4K	$1.0x10^{-2}$	-	-
	$0.5x10^{-3}$	-	-
	$1.0x10^{-3}$	-	-
8K	$1.0x10^{-2}$	-	-
	$0.5x10^{-3}$	-	-
	$1.0x10^{-3}$	-	-
12K	$1.0x10^{-2}$	-	-
	$0.5x10^{-3}$	-	-
	$1.0x10^{-3}$	-	-

- - : No error found till 1 million blocks.

First error till 1 million blocks

$n \simeq$	BER(In)$\simeq$	R=0.85	R=0.9	R=0.95
4K	$1.0 x 10^{-2}$	$2.677 x 10^{3}$	$1.7819 x 10^{4}$	1
	$0.5 x 10^{-3}$	$2.4944 x 10^{4}$	$1.65511 x 10^{5}$	$1.79 x 10^{2}$
	$1.0 x 10^{-3}$	$5.47550 x 10^{5}$	$4.89654 x 10^{5}$	$3.328 x 10^{3}$
8K	$1.0 x 10^{-2}$	$2.3817 x 10^{4}$	$1.16847 x 10^{5}$	1
	$0.5 x 10^{-3}$	$6.9491 x 10^{4}$	$1.72263 x 10^{5}$	$1.001 x 10^{3}$
	$1.0 x 10^{-3}$	$9.16505\ x 10^{5}$	$6.28939 x 10^{5}$	$9.338 x 10^{3}$
12K	$1.0 x 10^{-2}$	$9.705 x 10^{3}$	$5.37754 x 10^{5}$	1
	$0.5 x 10^{-3}$	$5.6400 x 10^{4}$	-	$1.318 x 10^{3}$
	$1.0 x 10^{-3}$	-	-	$1.6920 x 10^{4}$

- - : No error found till 1 million blocks.

Serial Min Sum Decoder - Random Matrix

- Min sum algorithm is implemented for Gaussian channel.
- Random matrix of block size(n=) 4K, 8K and 12K are formed using Mackey's algorithm.
- Five different code rates(R=) 0.75, 0.80, 0.85, 0.90 and 0.95 are taken.
- Raw input bit error rate(BER(IN)) is between 10^{-2} to 10^{-3}, converted in form of Eb/No(db) to express input SNR in db.
- BER(OUT) : Output bit error rate.
- We have tabulated when the first block get wrongly decoded till 1 million transmitted blocks.

First error in 1 million blocks

n≃	BER(In)≃	R=0.75	R=0.8
4K	1.0×10^{-2}	1.2799×10^{4}	2.0754×10^{4}
	0.5×10^{-3}	5.53727×10^{5}	1.72781×10^{5}
	1.0×10^{-3}	-	6.24436×10^{5}
8K	1.0×10^{-2}	1.92476×10^{5}	8.3898×10^{4}
	0.5×10^{-3}	3.21027×10^{5}	4.6092×10^{4}
	1.0×10^{-3}	-	-
12K	1.0×10^{-2}	2.20022×10^{5}	1.57371×10^{5}
	0.5×10^{-3}	2.17452×10^{5}	9.0158×10^{4}
	1.0×10^{-3}	-	-

- - : No error found till 1 million blocks.

First error in 1 million blocks

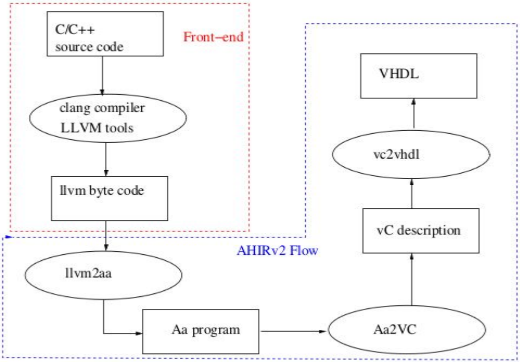
n≃	BER(In)≃	R=0.85	R=0.9	R=0.95
4K	1.0×10^{-2}	3.39×10^{2}	1.259×10^{3}	NA
	0.5×10^{-3}	6.6700×10^{4}	1.65511×10^{5}	NA
	1.0×10^{-3}	3.45503×10^{5}	1.19008×10^{5}	NA
8K	1.0×10^{-2}	5.193×10^{3}	5.947×10^{3}	NA
	0.5×10^{-3}	3.7952×10^{4}	1.1389×10^{4}	NA
	1.0×10^{-3}	-	-	NA
12K	1.0×10^{-2}	1.2894×10^{4}	1.2626×10^{4}	1.1
	0.5×10^{-3}	1.56487×10^{5}	5.4866×10^{4}	1.034×10^{4}
	1.0×10^{-3}	-	-	1.4759×10^{4}

- NA : Not Applicable. (Matrix was not formed)
- - : No error found till 1 million blocks.

Aa to VHDL -AHIR Tool Chain[4]

[4] https://github.com/madhavPdesai/ahir/release/docs/pdf/Overview.pdf.

Aa to VHDL -AHIR Tool Chain[4]

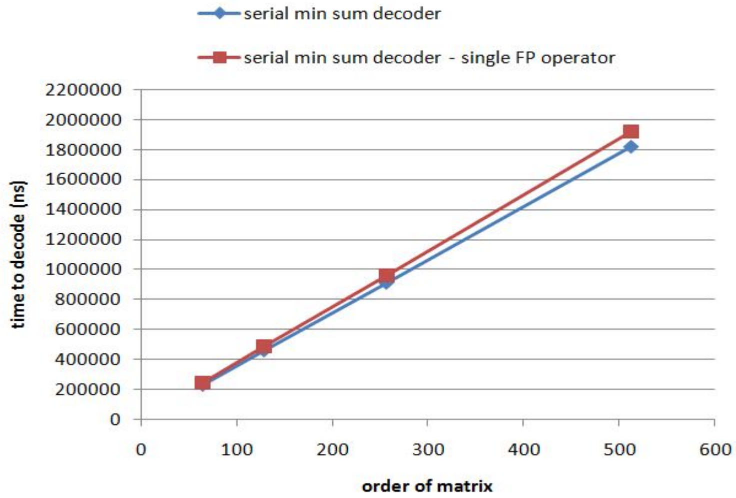


[4] https://github.com/madhavPdesai/ahir/release/docs/pdf/Overview.pdf.

Results


serial min sum decoder
serial min sum decoder - single FP operator
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
time to decode (ns)
64
128
256
512
order of matrix

Results


serial min sum decoder
serial min sum decoder - single FP operator
time to decode (ns)
2200000
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
0
100
200
300
400
500
600
order of matrix

Results : Hardware generated after synthesising the design

	serial min sum decoder	serial min sum decoder (single FP unit)
FF	18,076	19,034
LUT	19,502	20,621
Memory LUT	6	3
I/O	128	128
BRAM	56	56
BUFG	1	1

Partitioning : Objective

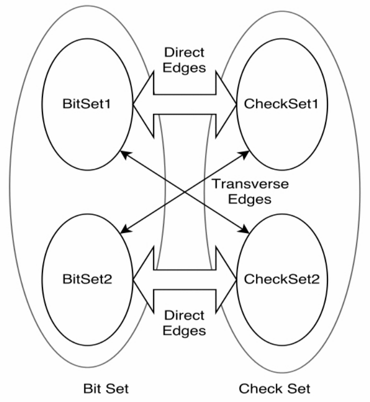


Figure: Partitioning the bipartite graph

Suggested Approach

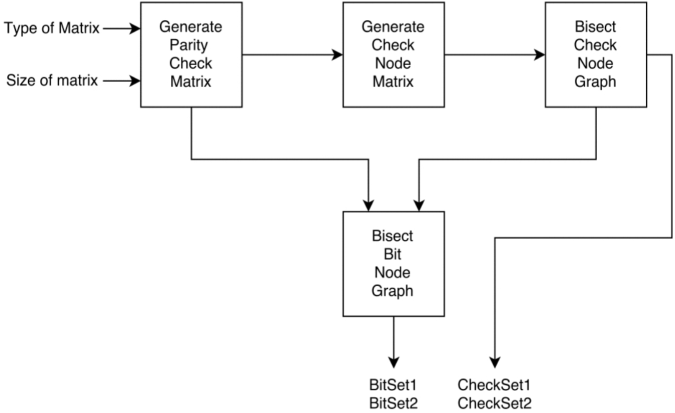


Figure: Block diagram

Partitioning Bit Node Set

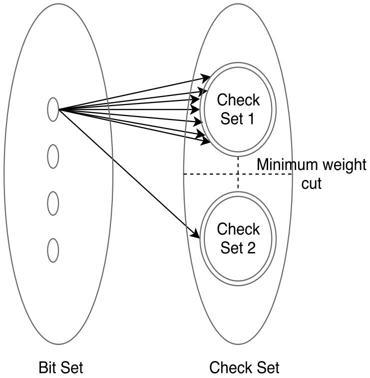


Figure: After partitioning check node set

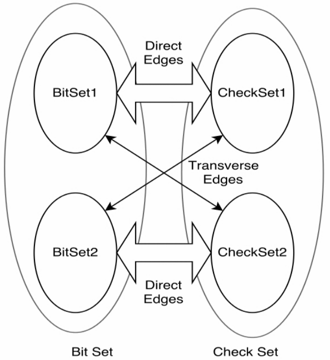
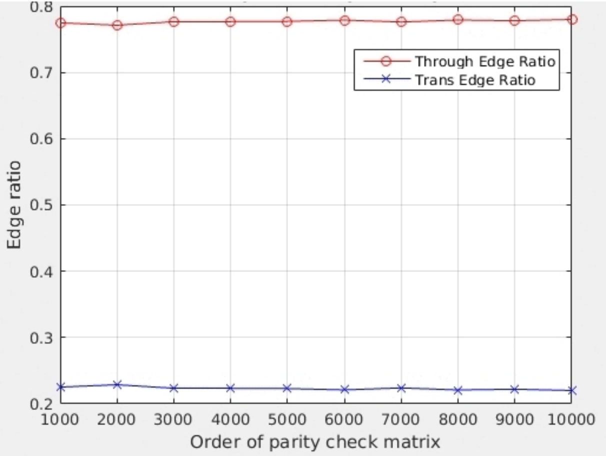
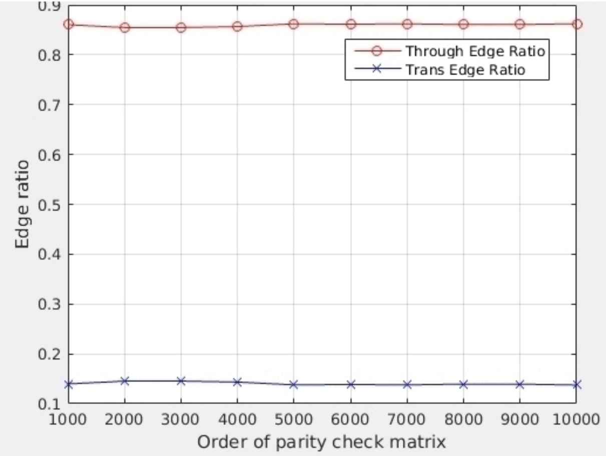


Figure: After partitioning bit node set

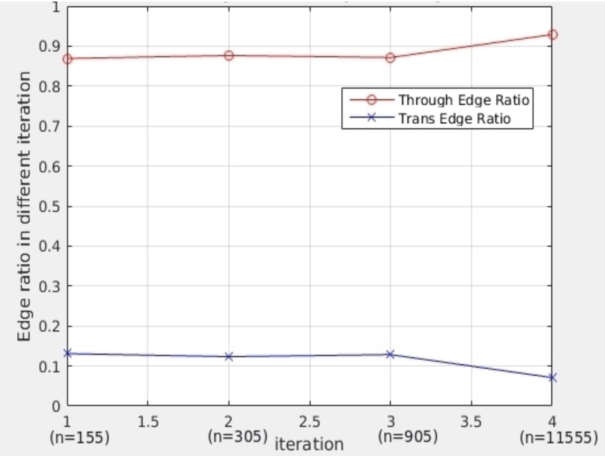
Gallager Parity Check Matrix



Mackay Parity Check Matrix



Quasi Cyclic Parity Check Matrix



Modifying min sum algorithm using partitioning

Modifying min sum algorithm using partitioning

- 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.

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

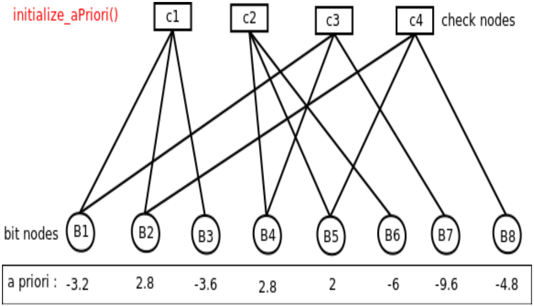
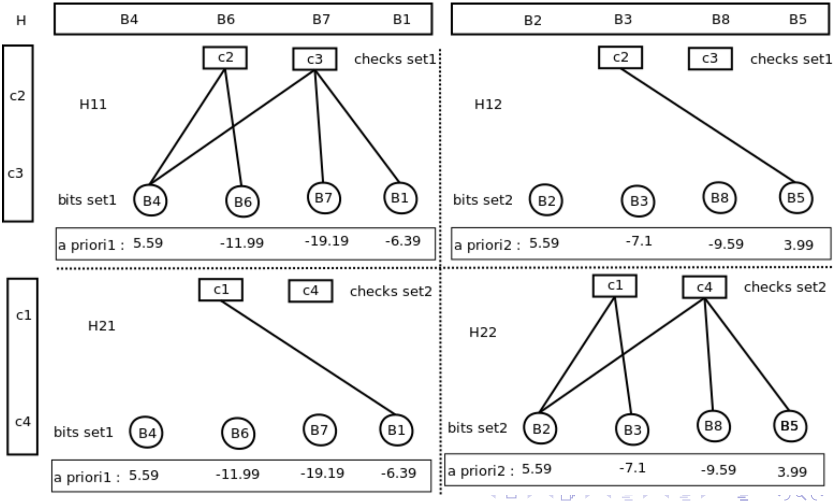


Figure: A priori initialization

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

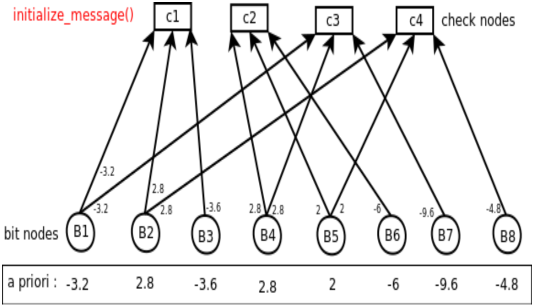
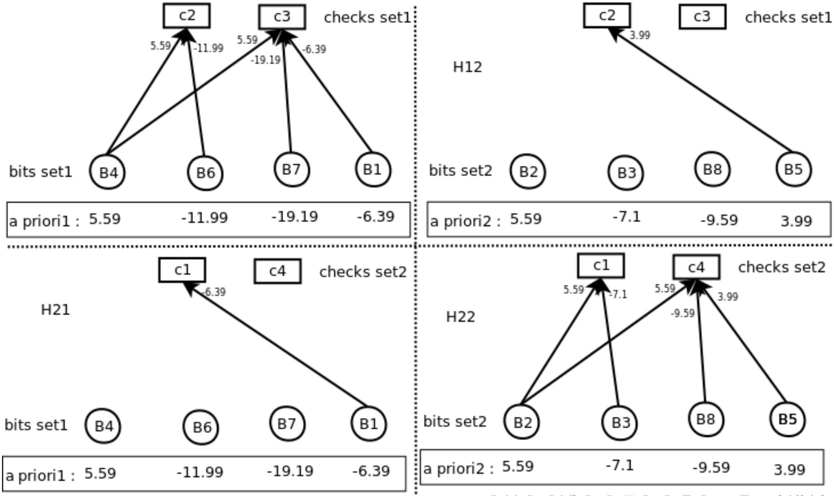


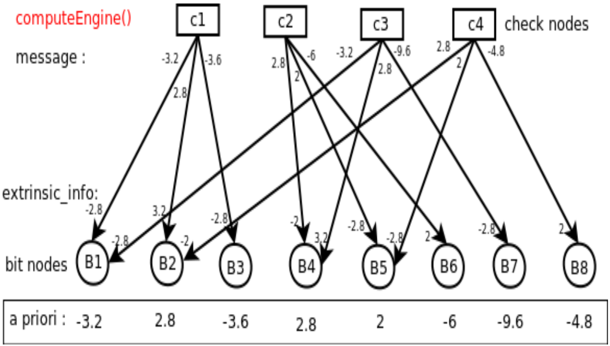
Figure: Message initialization

Message initialization



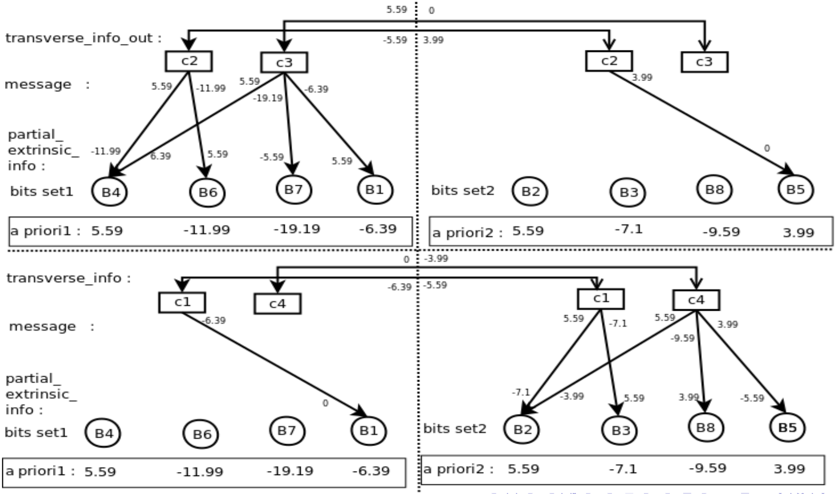
Extrinsic information calculation

- Extrinsic information of a bit node is calculated as min sum of all the messages connected to that particular check node.

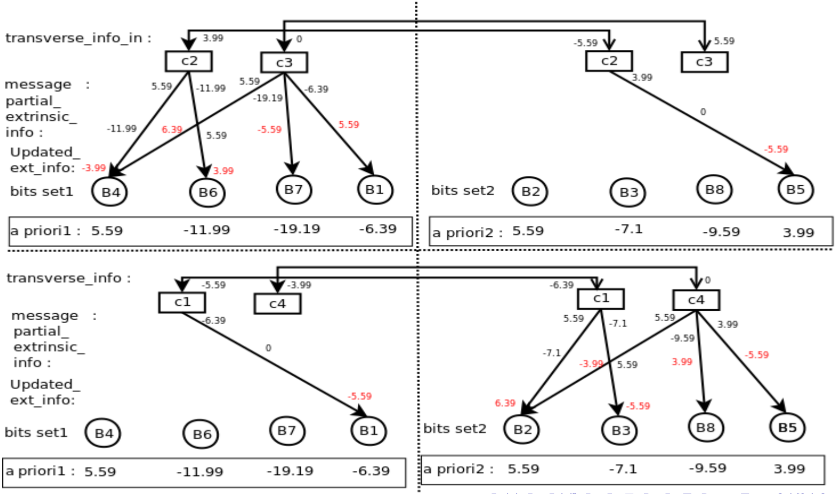


- $|E_{(j,i)}| = Min_{i' \in B_j \; i' \neq i} |M_{j,i'}|$
- $sign(E_{(j,i)}) = \prod_{i' \in B_j \; i' \neq i} sign(M_{j,i'})$

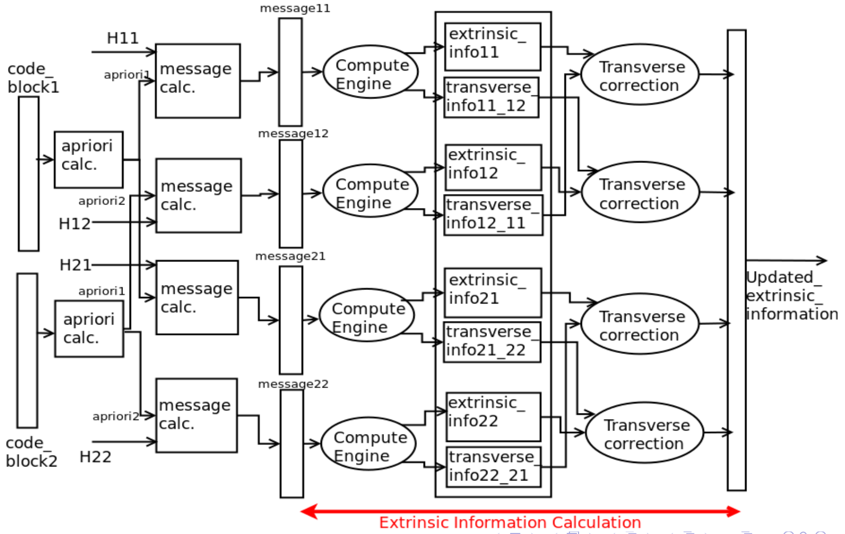
Partial Extrinsic information calculation



Update extrinsic information



C Level Implementation


code_
block1
code_
block2
H11
H12
H21
H22
apriori
calc.
apriori
calc.
apriori1
apriori2
apriori1
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

modifiedMinSumDecode() :

$initialize\_aPriori(aPriori1)$
$initialize\_aPriori(aPriori2)$
$initializeMessage(message11)$
$initializeMessage(message12)$
$initializeMessage(message21)$
$initializeMessage(message22)$
while $nitr \geq Max\_nitr$ **do**
  $initialize\_aPosteriori(aPosteriori1) \Leftarrow aPriori1$
  $initialize\_aPosteriori(aPosteriori2) \Leftarrow aPriori2$
  $initializeExtrinsicInfo(ext\_info11) \Leftarrow 0$
  $initializeExtrinsicInfo(ext\_info12) \Leftarrow 0$
  $initializeExtrinsicInfo(ext\_info21) \Leftarrow 0$
  $initializeExtrinsicInfo(ext\_info22) \Leftarrow 0$

  ...

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

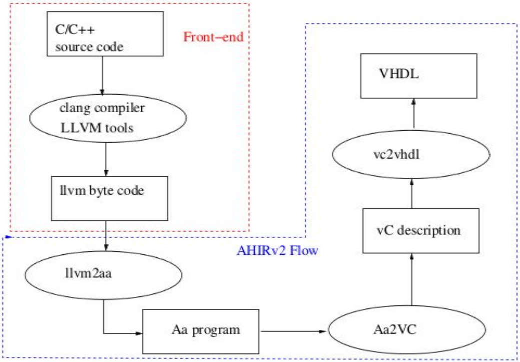
$nitr++$

end while

Aa to VHDL -AHIR Tool Chain[5]

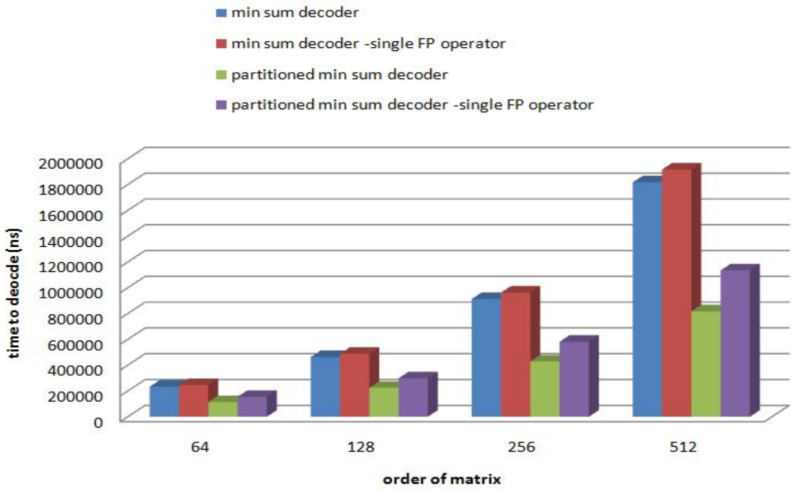
[5] https://github.com/madhavPdesai/ahir/release/docs/pdf/Overview.pdf.

Aa to VHDL -AHIR Tool Chain[5]

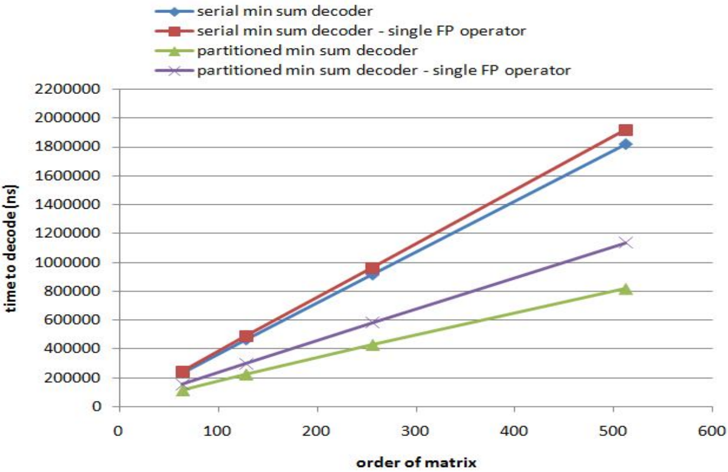


[5] https://github.com/madhavPdesai/ahir/release/docs/pdf/Overview.pdf.

Results


min sum decoder
min sum decoder -single FP operator
partitioned min sum decoder
partitioned min sum decoder -single FP operator
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
time to deocde (ns)
64
128
256
512
order of matrix

Results


serial min sum decoder
serial min sum decoder - single FP operator
partitioned min sum decoder
partitioned min sum decoder - single FP operator
time to decode (ns)
2200000
2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
0
100
200
300
400
500
600
order of matrix

Results

	serial min sum decoder	serial min sum decoder (single FP unit)	partitioned min sum decoder	partitioned min sum decoder(single FP unit)
FF	18,076	19,034	49,854	55,988
LUT	19,502	20,621	51,929	60,296
Memory LUT	6	3	23	2
I/O	128	128	128	128
BRAM	56	56	80	80
BUFG	1	1	1	1

Conclusion & Future Work

Type of Matrix	Gallager	Mackay Neal	Quasi-Cyclic
Performance Index	78%	86%	88%

- The results show that a LDPC decoder can be parallelized with good efficiency.
- The implemented 4-way partitioned decoder reduces the time required for decoding to half but uses 2.5× the harder required for single decoder.
- In future we can figure out a way to fold two engines on the top of other two engines to reduce hardware, instead of using four computational engines.
- The extension of the work can have different quantization levels of the floating point values and check for the trade off between accuracy of operation and error correcting threshold.

Thank You

Questions?