Some Aspects of Hardware Implementation of LDPC Codes

Emmanuel Boutillon
Fréderic Guilloud (PhD, ENST)
Jean-Luc Danger (ENST)

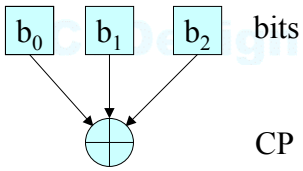
Laboratoire d'électronique des systèmes temps réel
Université de Bretagne Sud
UPRES EA 3372
emmanuel.boutillon@univ-ubs.fr

Outline

- Definition of LDPC Code
  - Decoding of a Parity Check
  - Construction of a LDPC code
  - Performances
- Architecture of LDPC decoder
- Serial decoding of parity check
- Simulation results
- Conclusion

Parity check

Parity check (3,2,1) :

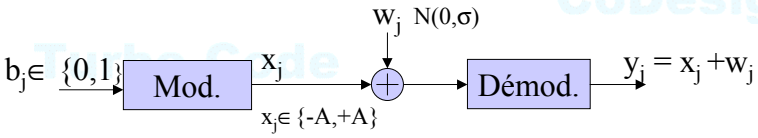


(b_0, b_1, b_2) codeword <=> Sum of 1 = 0 mod 2.

CP(3,2,1) = {(0, 0, 0) ; (0, 1, 1) ; (1, 1, 0) ; (1, 0, 1)}

Generalization: Parity Check (n, n-1, 1)

Decoding a PC



The observation y_j gives the *intrinsic* information i_j of bit b_j:

$i_j = (p(b_j=0/y_j) , p(b_j=1/y_j))$

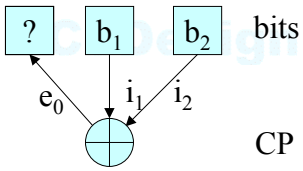
or in an more convenient representation (log likelihood ratio):

$i_j = \ln(p(b_j=1/y_j)/p(b_j=0/y_j)) = 2y_j/\sigma^2$

Note: sign(i_j) => hard decision, | i_j | reliability of the decision

Decoding a parity check code

What information i_1 and i_2 gives over the value of b_0 ?



Using CP

$$p(b_0=0/y_1,y_2) = p(b_1=0/y_1).\ p(b_2=0/y_2) + p(b_1=1/y_1)\ .\ p(b_2=1/y_2)$$
$$p(b_0=1/y_1,y_2) = p(b_1=0/y_1).\ p(b_2=1/y_2) + p(b_1=1/y_1)\ .\ p(b_2=0/y_2)$$

we have and extra independent information e_0 over b_0 given by:

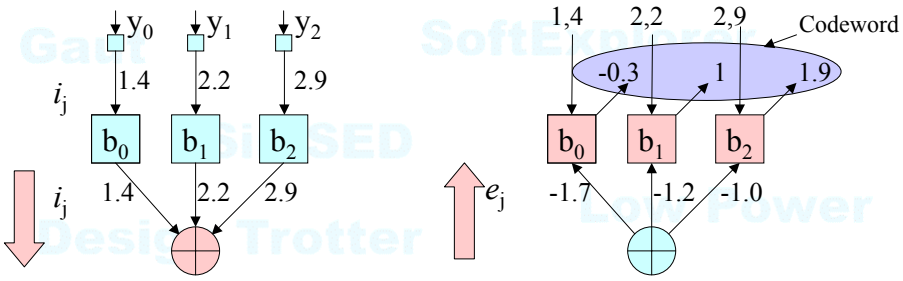
$$e_0 = i_1 \oplus i_2 = \ln\left(\frac{1+e^{i_1}\cdot e^{i_2}}{e^{i_1}+e^{i_2}}\right)$$

e_0 is named the *extrinsic* information.

Decoding a parity check code

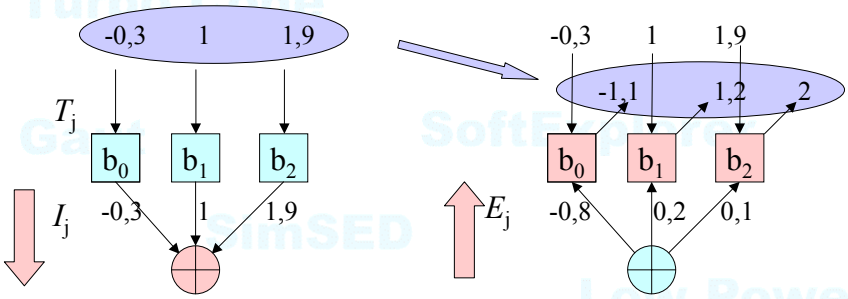
Then i_0 and e_0 are added to obtain the final decoding

The process is symmetrical for all bits



Note: iteration

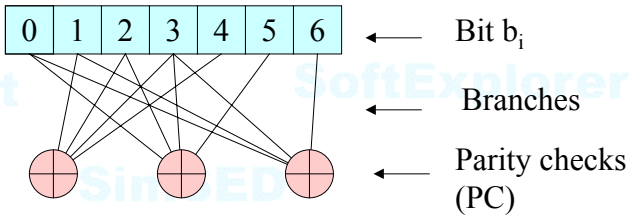
If the process is re-iterate, there is auto-confirmation



and the system diverge...

LDPC Code

It can be defined by a bi-partite graph



(b_0, b_1, … , b_6) is a codeword <=> all PC are respected.

Why LDPC ?

Algebraic representation:

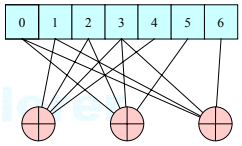
$X = (x_0, x_1, \ldots, x_6)^T$ is a codeword if $H.X = 0$

$$H = \begin{pmatrix} 0\,1\,1\,1\,1\,0\,0 \\ 1\,0\,1\,1\,0\,1\,0 \\ 1\,1\,0\,1\,0\,0\,1 \end{pmatrix}$$

P=number of PC

Check Matrix

N=number of bits



Number of 1 over a row = dc=number of bit associated to the PC
Number of 1 over a line = dv=number of PC associated to the bit

Less than 1% of the bits of H are equal to 1

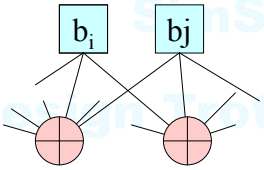
How to construct a good LDPC Code

Select first the size and the rate of the code

Select an optimal spectrum repartition of the branch

example:
90 % bits => dv =3 branches, 10 % bits dv=> 12 branches
70 % PC => dc = 6 branches, 30 % PC => dc = 8 branches

than chose the code randomly… just avoiding short cycle like:

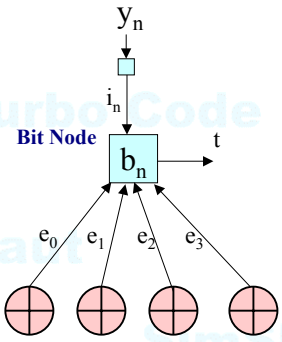


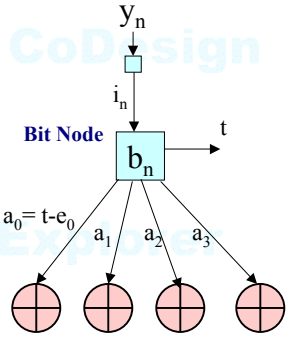
…and you have a good code

BP (or sum-product) decoding algorithm

- Step1: compute the LLR of received bit
- Step2: Message passing from bit to check + check processing.
- Step3: Message passing from check to bit + bit processing.
- Step4: repeat 2 and 3 until decoding OK or "max iteration" obtained.

Processing in bit node

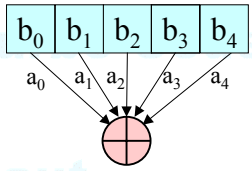




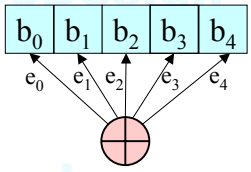
$$t = i_n + \sum_m e_m$$

$$a_i = i_n + \sum_{m,m\neq i} e_i = t - a_i$$

Processing in check nodes : principles



Check Node

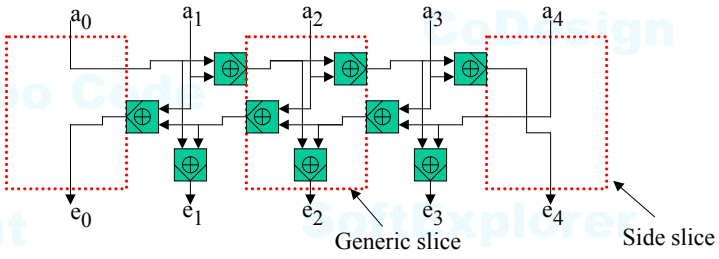


Check Node

$$e_i = \bigoplus_{j \neq i} a_i = a_1 \oplus a_2 \ldots \oplus a_{i-1} \oplus a_{i+1} \ldots \oplus a_{dc}$$

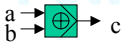
No simplifications...

Processing in check node: realization



G.D. Forney « on iterative decoding and the two-way algorithm » Symp. On Turbo-Codes, Brest 1997

Complexity: 3.(dc - 2) operator "XOR"

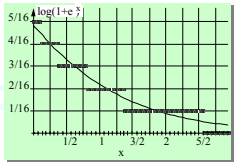


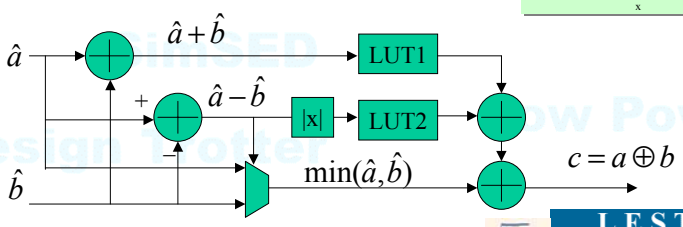
Realization of "XOR"

let $s_a = sign(a), s_b = sign(b), \hat{a} = |a|, \hat{b} = |b|$

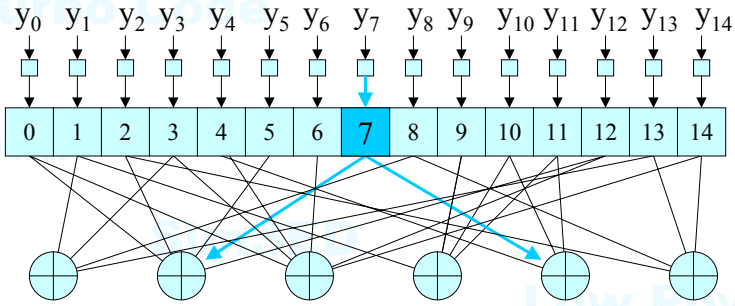
$$c = a \oplus b = \ln\left(\frac{1 + e^a \cdot e^b}{e^a + e^b}\right) = -s_a . s_b . \left(\min(\hat{a}, \hat{b}) - F\left(|\hat{a} - \hat{b}|\right) + F(\hat{a} + \hat{b})\right)$$

With $F(x) = \ln(1 + e^{-x})$ LUT



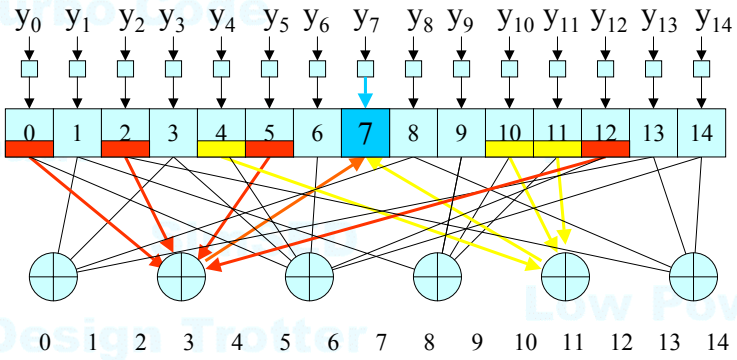


Iterative decoding



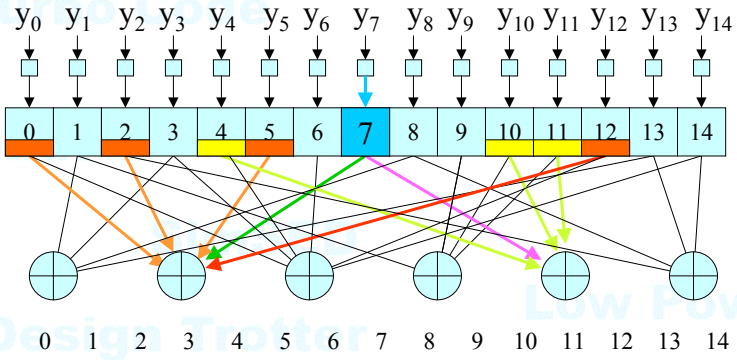
n decoding iterations

Iterative decoding



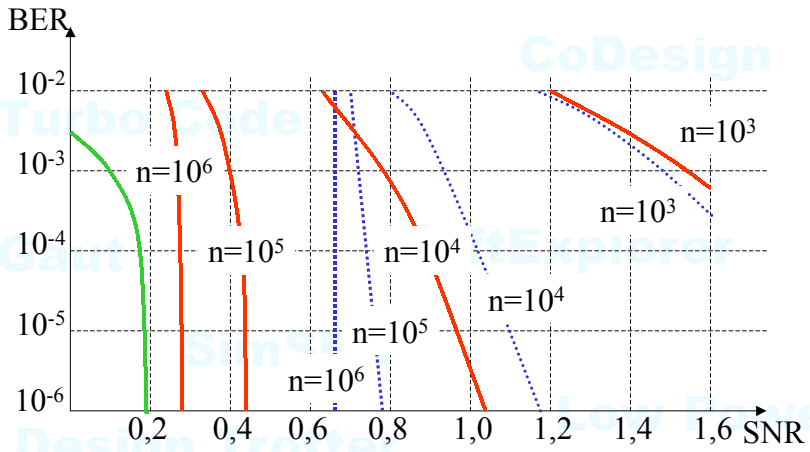
n decoding iterations

Iterative decoding



n decoding iterations

Performances (rate 1/2)



Best known code for $n > 10^4$

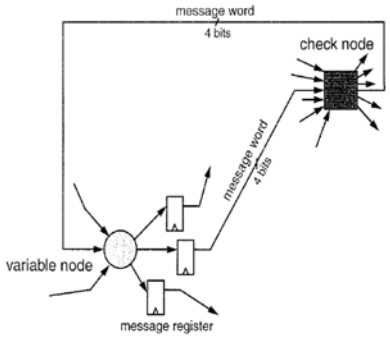
Outline

- Definition of LDPC Code
- Architecture of LDPC decoder
  - Full parallel decoder
  - Decoder first code design
- Serial decoding of parity check
- Simulation results
- Conclusion

Full parallel architecture

- Example

  Blanksby, Howland, « *A 690-mW 1-Gb/s 1024-b Rate-1/2 Low-Density Parity-Check Code Decoder* » (IEEE Trans. on Solid-State Circuits, 2002.)



- Avantages
  - performance: 1Gb/s 64 iterations
  - Power dissipation : 690mW
  - PER=2 10^{-4} @ 2,5 dB
- Drawback
  - Complex routing =>add oc CAD tool
  - Fixed code
  - Size : 52.5mm^2 , 0.16µ Techno

Direct implementation

Serial-parallel architecture

Main problem: access to the memory

Example: if Fclock = Fbit, then, at each clock cycle the number A of read/write memory access is given by:

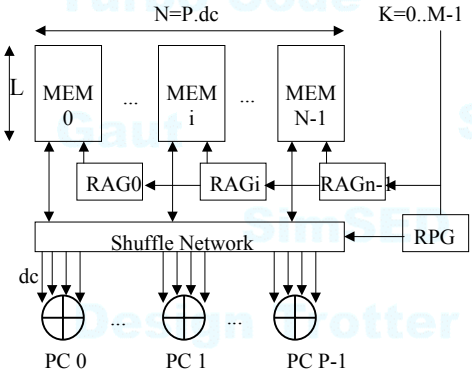
Where dc is the average degree of PC
r is the rate of the code
nb_{iter} is the number of iteration

Example: dc=6, r=1/2 and nb_{iter} = 50 give $A = 150$

Solutions: a) Duplicate the memory
b) Split the memory in small blocks for parallel decoding

Method « Decoder first code design »

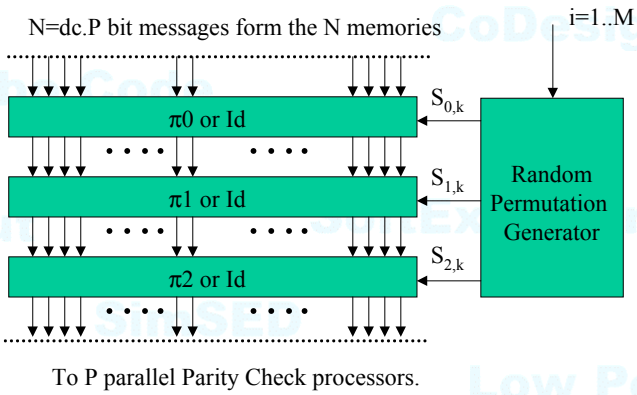
- Method proposed in Boutillon et al. « *Decoder-First Code Design* », Int. Symp. on Turbo Codes, Brest, 2000.



- The architecture is first defined
  - P Parity Check are done every cycle.
  - Memory is divided in N=P*dc sub_memory of size L.
- Code of size NL with MP parity check.
- Decoding iteration: M cycles.
  - Retrieve bit from memory at the address given by the Random Address Generator (RAG).
  - Shuffle the data(Random Permutation Generator).
  - Perform the PC process.
  - Unshuffle the extrinsic information serial on fly process of bit nodes.

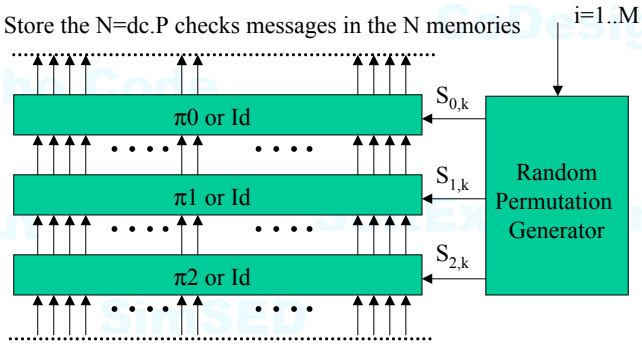
No performance degradations...

Shuffle Network: from node to PC



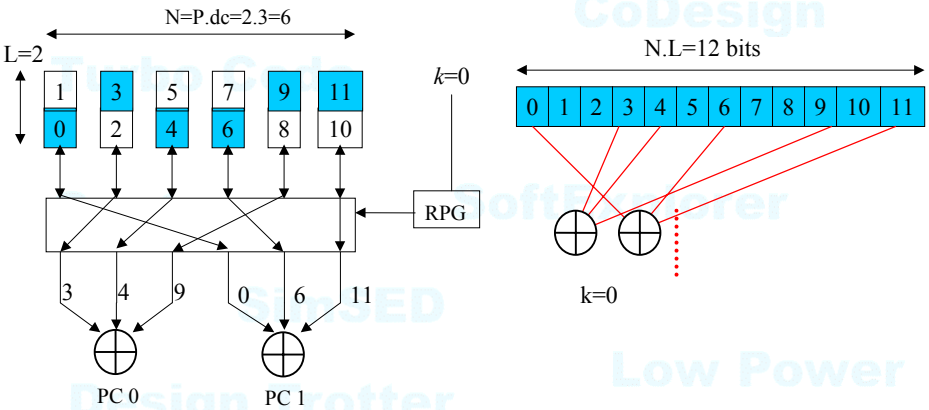
To P parallel Parity Check processors.

Shuffle Network: from PC to node

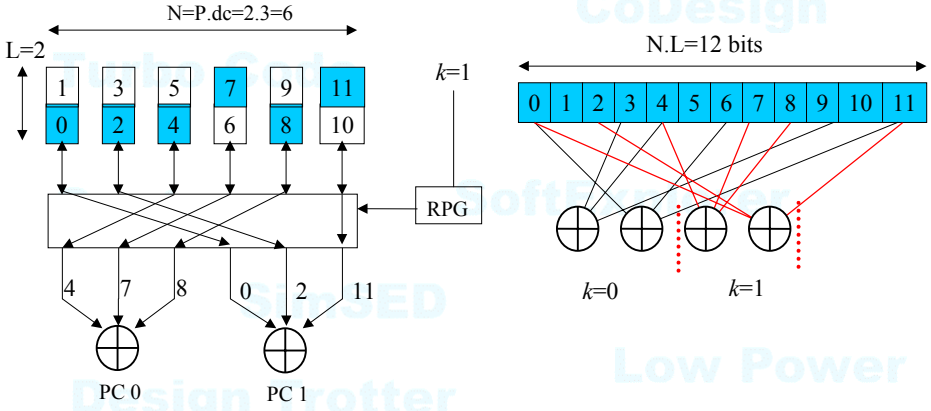


From P parallel Parity Check processors.

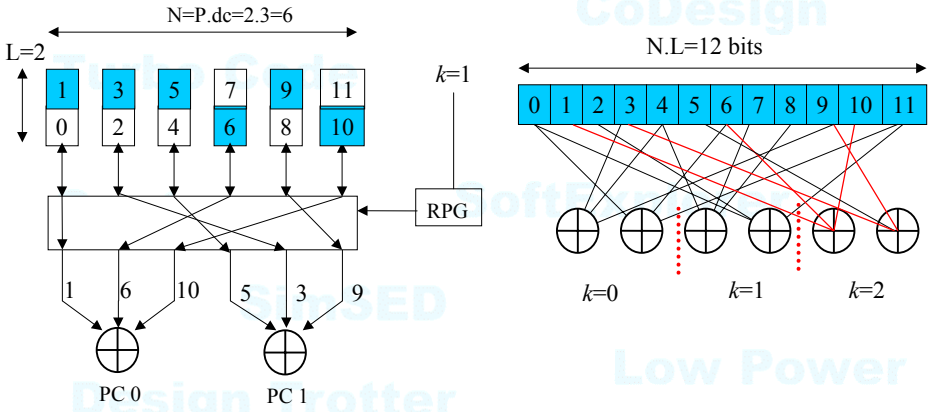
Example: k=1



Example: k=1

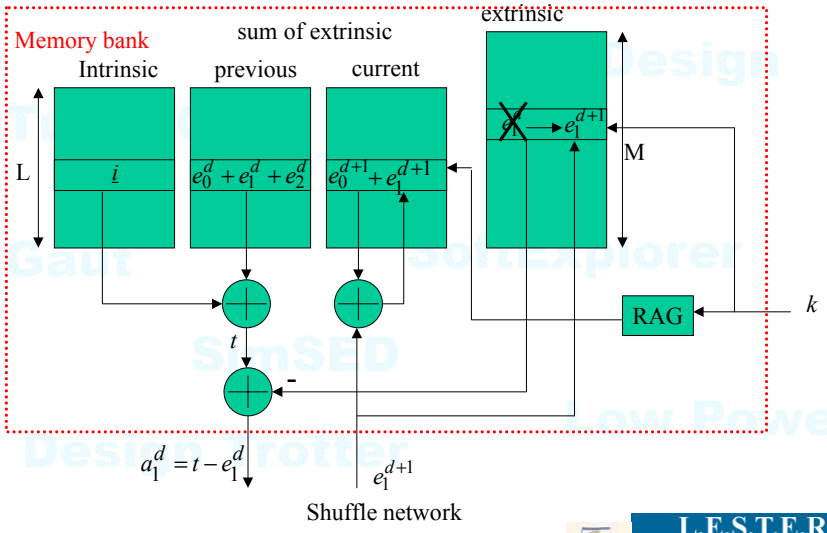


Example: k=2



One decoding iteration performed every M=3 cycles.

Memory: serial computation of bit node



Reported work on LDPC design

Parrallele architecture

- No constraint on the code (Blanksby, Howland)
- Constraint on the code (Sobelman)

Parrallele-serial architecture

- No constraint on the code (Lee, Wu)
- Constraint on the code (Verdier, Tong Zhang, Parhi)
  - "Joint code and decoder design for implementation-oriented (3, k)-regular LDPC codes", Tong Zhang; Parhi, K.K, Asilomar 2001.

◆ Industry

Flavion technology

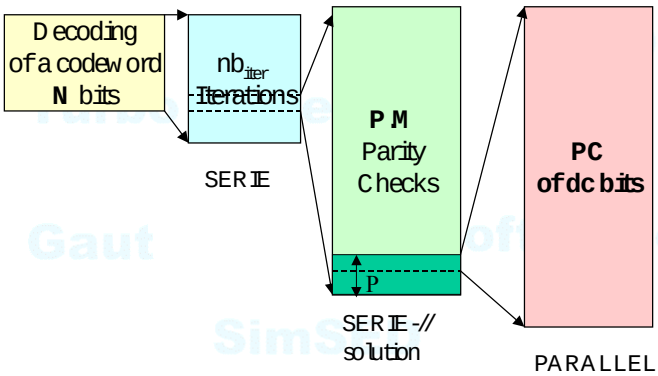
Hugues Network System

- Proposition for DVB-S2.

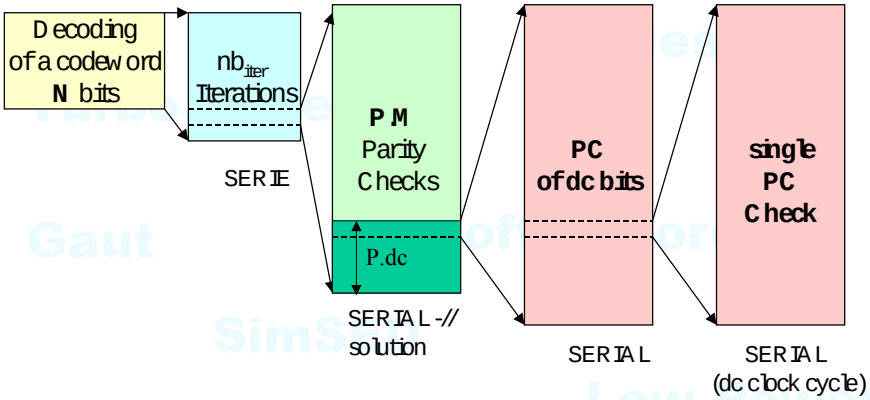
Outline

- Definition of LDPC Code
- Architecture of LDPC decoder
- Serial decoding of parity check
  - simplification of PC equations
  - architecture of PC processor
  - Reduction of extrinsic memory
- Simulation results
- Conclusion

Actual scheduling

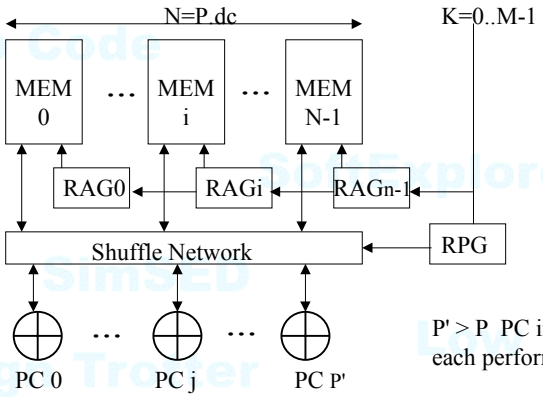


Proposed new scheduling

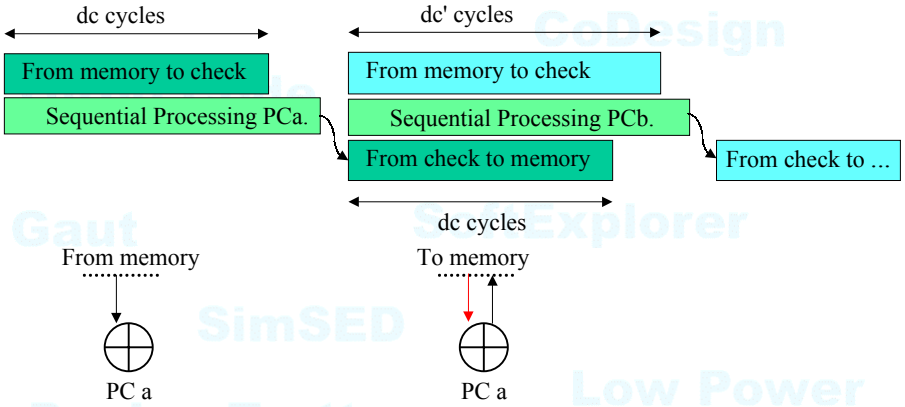


Objective: sub_optimal algorithm with reduced complexity

Serial architecture



Serial Processing of check node



Irregular LDPC are well managed

λ-min algorithm

Assuming $a_i = s_i.\hat{a}_i$ the expression of the e_i can be expressed by:

$$e_i = \bigoplus_{j,j\neq i} a_j = s_i \cdot S \bigoplus_{j,j\neq i} \hat{a}_j$$

Where $S = \prod_i - s_i$

Term dominated by the smallest values

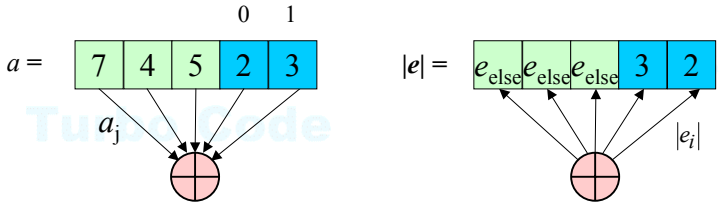
Idea: computed it only with the λ smallest values of $\hat{a}_j$

=> λ-min algorithm

$$|e_i| \approx \alpha \bigoplus_{j\in I, j\neq i} \hat{a}_j \qquad I = \{ind_0,\ldots,ind_{\lambda-1}\}$$

$$\hat{a}_{ind_0} \leq \ldots \leq \hat{a}_{ind_{\lambda-1}} \leq \hat{a}_k \qquad k \notin I$$

Example λ=2-min algorithm



$ind_0 = 3$
$ind_1 = 4$

$$e_3 = \bigoplus_{j\in\{3,4\}/\, j\neq 3} a_j = a_4 = 3$$

$$e_4 = \bigoplus_{j\in\{3,4\}/\, j\neq 4} a_j = a_3 = 2$$

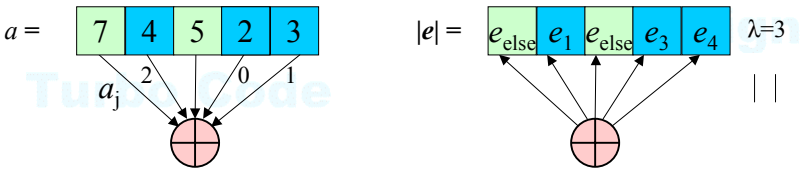
$$e_{else} = \bigoplus_{j\in\{3,4\}} a_j = a_3 \oplus a_4$$

Note: Fossorier proposed independantly

$$e_j = \min_{i\,/\, j\neq i} a_j$$

+ correction factor

Example λ=3-min algorithm



$ind_0 = 3$
$ind_1 = 4$
$ind_2 = 1$

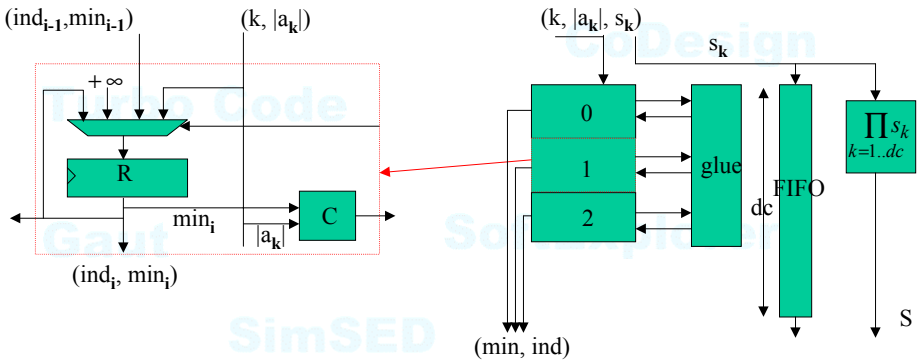
$e_i =$ ⊕ a_j

$e_{else} = a_3 \oplus a_4 \oplus a_1$

$e_1 = a_3 \oplus a_4$

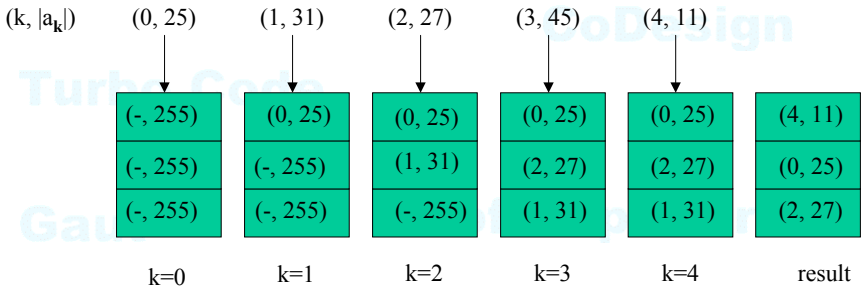
$e_3 = a_4 \oplus a_1$

Serial sorting : 1) synthesis



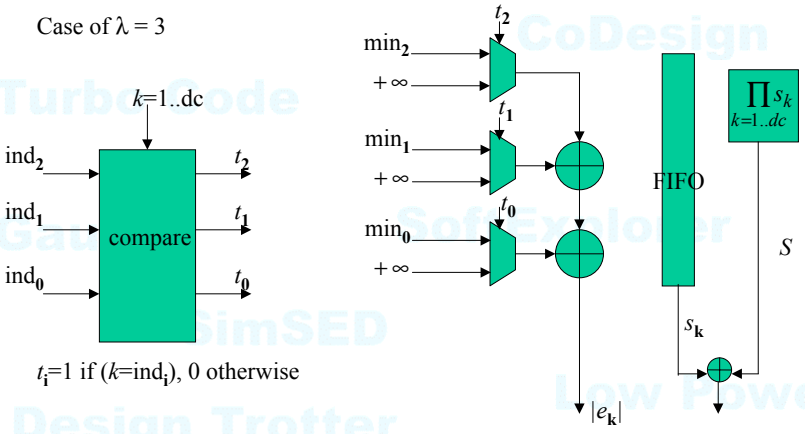
After the serial arrival of the dc bit message
the λ mins are known.

Serial sorting : Example for dc=5, λ=3

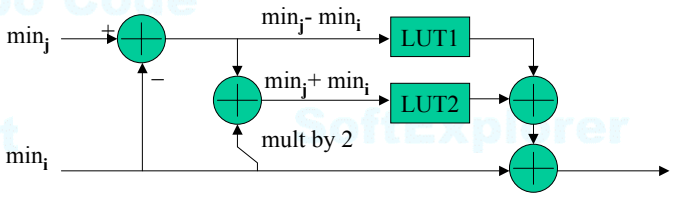


After the serial arrival of the dc bit message
the λ min are known.

"On the fly" parity check scheme

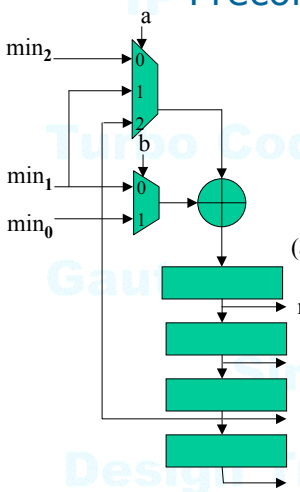


Note: XOR function is simplified



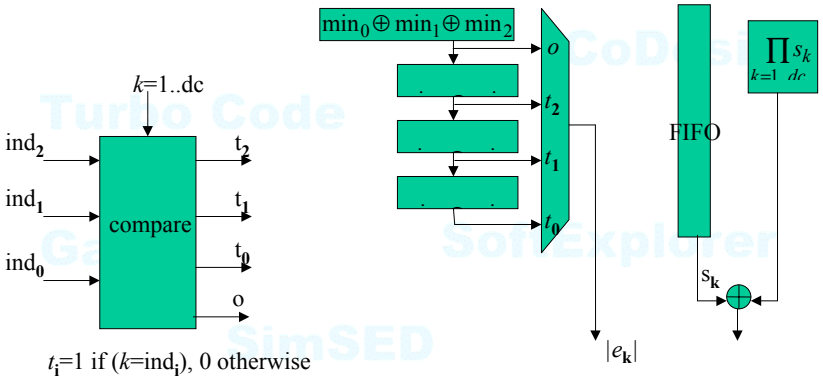
PC operator can be simplified using $min_j > min_i$

Precomputed PC: Computing



cycle 1	cycle 2	cycle 3	cycle 4
(a,b)=(0,0)	(a,b)=(0,1)	(a,b)=(1,1)	(a,b)=(1,2)
$\min_1 \oplus \min_2$	$\min_0 \oplus \min_2$	$\min_0 \oplus \min_1$	$\min_0 \oplus \min_1 \oplus \min_2$
	$\min_1 \oplus \min_2$	$\min_0 \oplus \min_2$	$\min_0 \oplus \min_1$
		$\min_1 \oplus \min_2$	$\min_0 \oplus \min_2$
			$\min_1 \oplus \min_2$

Precomputed PC scheme: extrinsic generation



t_i=1 if (k=ind$_i$), 0 otherwise

o = 1 if t_1=t_2=t_3=0

IDEA : Instead of storing extrinsic, store just the information needed to re-generate it

Storage of extrinsic information

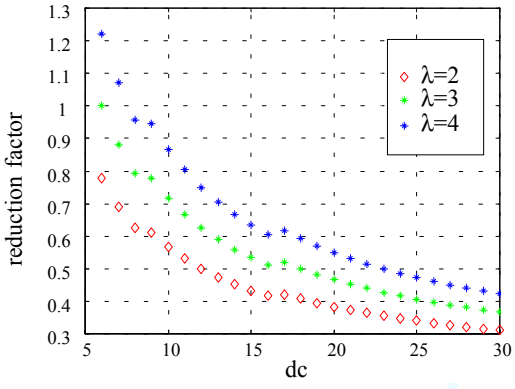
Classical method: each PC needs the storage of dc extrinsic values
=> dc.(1+N_b) bit of memory

Proposed method: store the information needed to recompute extrinsic values instead of storing extrinsic values

Information to be stored:	Number of bits
λ+1 min values →	(λ+1).N_b
λ indice values →	λ $\log_2$(dc)
dc signs →	dc bits
Product of sign →	1 bits

The compression ratio is: $\frac{1+dc+\lambda\log_2(dc)+(\lambda+1)N_b}{dc.(1+N_b)}$

Result

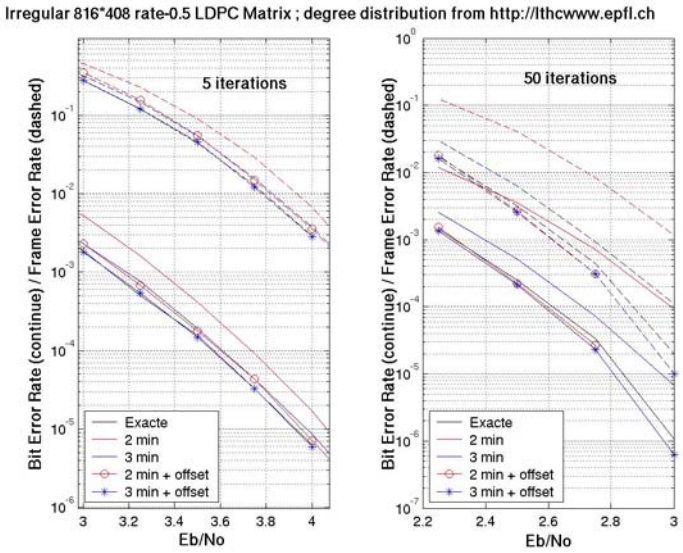


Trade-off "memory save" <-> "Computation"

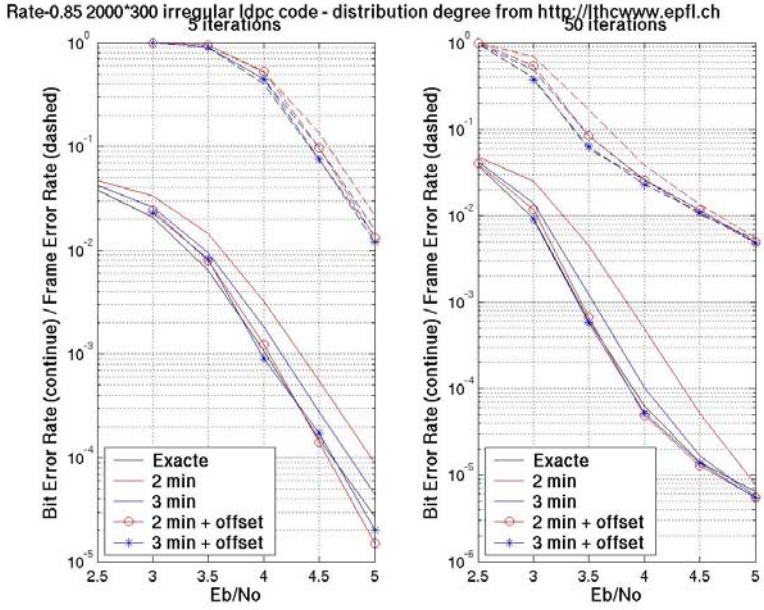
Outline

- Definition of LDPC Code
- Architecture of LDPC decoder
- Serial decoding of parity check
- Simulation results
  - Rate 1/2
  - MDPC
- Conclusion

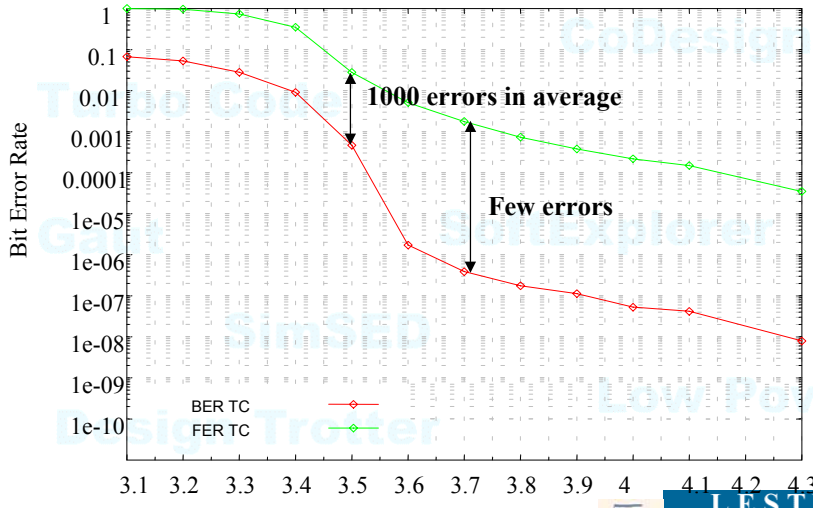
Some result



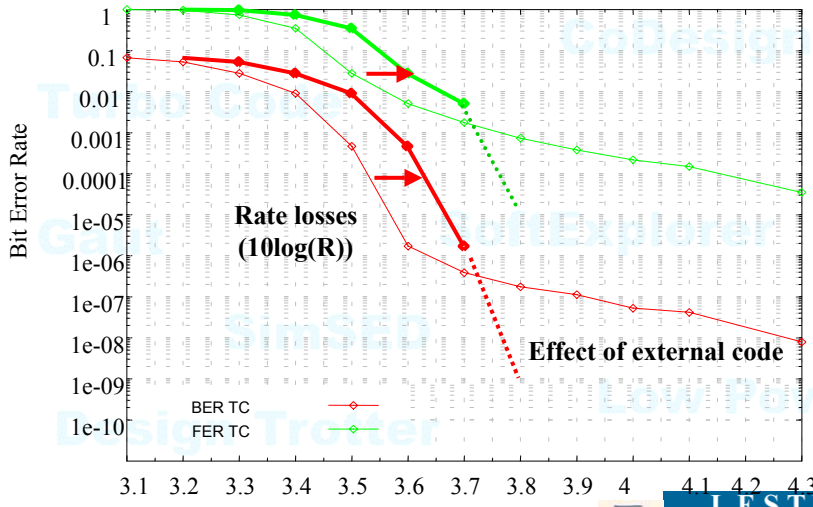
Offset β : extrinsic = max(0, *e* - β)



Performances of high rate Turbo-Code



Use an external code to suppress the flatening



Construction of MDPC

External code :

- LDPC with few hundred of bit per parity (=> Medium DPC)
- very high coding rate (arround 0,98).

At the output of the Turbo-Code, most of the bit can be assumed correctly decoded.

=> they can be supressed of the MDPC code

After suppression of reliable bit of the MDPC code:

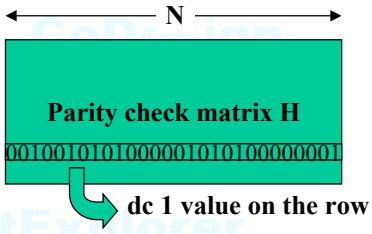
=> the code becomes LDPC

=> the coding rate is low

The decoding starts to be efficient

Construction of the Parity Check Matrix

N length of the code
P number of parity check
dc_i is the degre of the parity check P_i
dv_i is the degre of the bit b_i
$F(b_i)$ is the set of PC connected to b_i
$G(P_i)$ is the set of bit connected to PC P_i



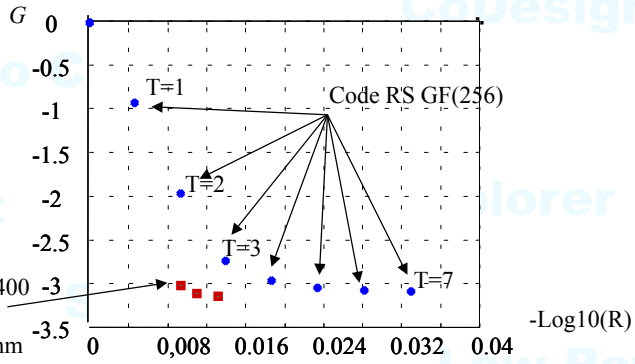
Constraint on the code to obtain a regular MDPC:

dv_i= constant for all i

$\forall i, i \neq j$ $Card(G(P_i) \cap G(P_j)) \leq Max$

with Max as minimum as possible

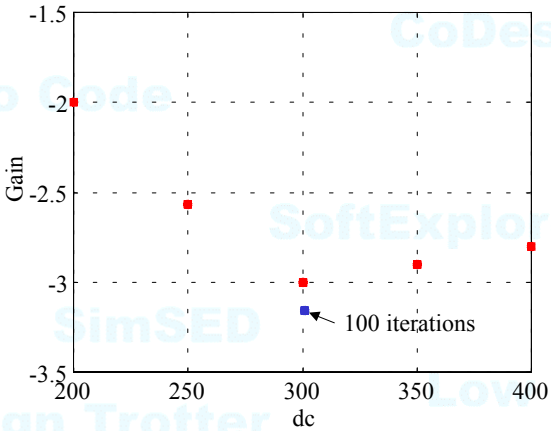
Results (regular MDPC)



Code MDPC
P = 320, 350 et 400
dc = 300
λ=2 min algorithm
10 iterations

Compared to RS code, the proposed scheme is more efficient

Influence of parameter dc



P = 320

10 decoding iterations.

There is an optimal value of dc

Conclusion

- With the choice of LDPC as a standard for DVB-S2, VLSI architecture for LDPC becomes a real challenge.
- Proposition of λ-min algorithm to reduce the complexity
  - Very efficient method in case of high rate LDPC
  - Use Fossorier's method to reduce the sub-optimality of the code (offset and/or constant factor on extrinsic values).
- Ondoing implementation of the λ-min algorithm on FPGA
  - Nallatec system, virtex1000E, 50 MHz
  - P=8 up to 16 parallelism degree
  - code size up to 2K.