DECODER-FIRST CODE DESIGN

Emmanuel BOUTILLON*, Jeff CASTURA** and Frank R. KSCHISCHANG**

*ENST, 46 rue Barrault, 75634 Paris Cedex 13, France
**University of Toronto, 10 King's College Road, M5S 3G4, Toronto, Ontario
E-mail: emmanuel.boutillon@enst.fr, {fcastura,frank}@comm.toronto.edu

Abstract: *The natural approch for the design of an error correction system is first to construct a code, then, define the hardware structure of the decoder. Unfortunately, such a constructed code has very little chance to be suited for a hardware implementation. This paper proposes to operate the other way: in the first step an efficient hardware structure is chosen and in the second step, a code is constructed that adequately fits this structure.*

An example of such a methodology is exposed for a Low Density Parity Check code. A cascadable high speed (clock cycle equal to bit rate) decoder archictecture is presented and simulation results are given. For an N=4096 LDPC code of rate 1/2, a Bit Error Rate of 10^{-4} *is acheived respectively for SNRs of 2.63 dB (1 circuit), 2.3 dB (2 circuits) and 2.05 dB (10 circuits).*

Keywords: LDPC, VLSI, architecture, code, design.

1. INTRODUCTION

Shannon theory tells us that all "random" codes are "good codes". Unfortunately, "random" codes are not likely to be practically decodable. Indeed, for many years, people in the coding theory community operated under the assumption that "all codes are good, except the ones we can imagine," (or more accurately, "except the ones we can imagine decoding"). However, thanks to the work of Gallager [1], Berrou [2] and others, we know, nowdays, that good codes can not only be constructed, but can also be decoded efficiently at signal-to-noise ratios very nearly approaching the Shannon limit.

Such codes are often represented geometrically by a bipartite graph, sometimes called a Tanner graph or factor graph, in which the two vertex classes represent the codeword symbols (and hidden state variables) on the one hand, and codeword "checks" on the other. Codeword checks might constrain neighbouring variables to form a codeword in a linear "constituent" code. Usually a "random-like" interconnection network between the two rows allows one to create good "random-like" codes. Such codes can be decoded with an iterative decoding strategy, sometimes with astounding performance.

Turbo codes can be thought of as an extreme case of this family of codes with just two complicated check vertices, constraining certain subsets of the variables to form a convolutional codeword. Due to the coarse granularity of the check vertices, the hardware realisation of the decoder is straightforward, since the interconnection network of the factor graph leads to a temporal shuffling of bits through an interleaver [3].

At the other end of the spectrum are low-density parity-check (LDPC) codes, where each check vertex is extremely simple: it constrains the neighbouring variables to form a word of even parity (i.e., a codeword in a simple parity-check code). Recent results [4,5] show that this type of code can outperform turbo-codes. Moreover, LDPC codes have excellent Hamming distance properties, which makes it unlikely for the decoder to converge to a wrong codeword.

At the present time, part of the coding theory community is working to improve the performance of this type of code. The "Holy Grail" is to construct the "best" code, but little consideration is given to its hardware implementation. Unfortunately, due to the large number of check vertices in the factor graph, the realization of a high speed decoder implies not only temporal shuffling, but also spatial shuffling of the data in order to emulate the interconnection network of the factor graph. Thus, the question arises about whether the LPDC can be suitable for high speed hardware realizations that are required in many communication links.

This paper proposes to reverse the sequence order of conventional approach to code design. In a first step a suitable hardware structure is chosen and in the second step, a code is constructed that adequately fits this structure. Thus, by construction, the code is suitable for VLSI implementation. The target, in this paper, is high speed LDPC decoders for which the input symbol and clock rates are equal.

2. DESCRIPTION OF THE METHOD

The general architecture of the hardware is defined prior to the code construction, as said above. The general architecture is first presented, then the sequencing of the computation is explained. Finally, sub-blocks are described.

2.1. Architecture overview

As shown is figure 1, the architecture is based on N memory banks of length L (for the $N.L$ codeword variables) ; an interconnection network for the spatial shuffling, and an array of P parity check decoders. The randomness of the code is obtained with the N Random Address Generators (RAG) associated to the memory banks and a Random Permutation Generator (RPG) that

configure the interconnection network. The P check nodes are connected to the output of the shuffle network.

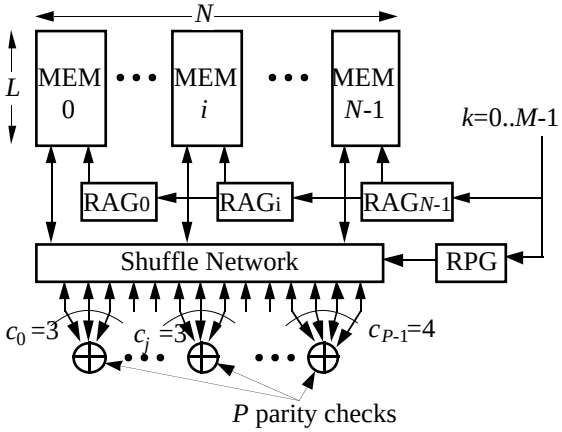

Figure 1: Principle of the architecture

The first check node is connected to the first (from left to right) c_0 output of the shuffle network, the second to the next c_1 output and so on. The sum of the $\{c_i\}_{i=0..P-1}$ values is thus equal to N.

2.2. Decoding process

The decoding process, as well as the architecture, is defined prior to the code construction. It is composed of D decoding iterations. Each decoding iteration is processed in M clock cycles. At every clock cycle, the 5 following operations are performed (eventually, with some pipeline):

- retrieve N data from the memory banks at the adresses given by the N RAG.
- shuffle the N data according to the RPG index.
- perform the MAP algorithm on the P parity checks. The result is the extrinsic information associated to each data.
- unshuffle the N extrinsic information computed by the parity checks.
- store the unshuffle data into the memory bank at theire initial location.

The code characteristics are derived from the hardware. parameters.

2.3. Code characteristic

The code length is $N.L$, and the number of parity checks of the code is $M.P$. Since each parity check corresponds to one bit of redundancy, the rate r of the code is:

$$r = \frac{NL-MP}{NL} \qquad (1)$$

From the throughput constraint (input symbol rate equal to the bit rate), the D codeword iterations should be performed in NL clock cycles. This constraint leads to:

$$D = \frac{NL}{M} \qquad (2)$$

The average numbers d_v of parity checks associated to each bit and the average number d_v of bits involved in each parity check are:

$$d_v = \frac{PM}{NL}\,;\; d_c = \frac{N}{P} \qquad (3)$$

Let us describe in more detail the different sub-blocks of the architecture.

2.4 Parity check

All the parity checks are factored [6] linearly in order to obtain a regular architecture, as shown in figure 2 for the case of a 4 bit parity check.

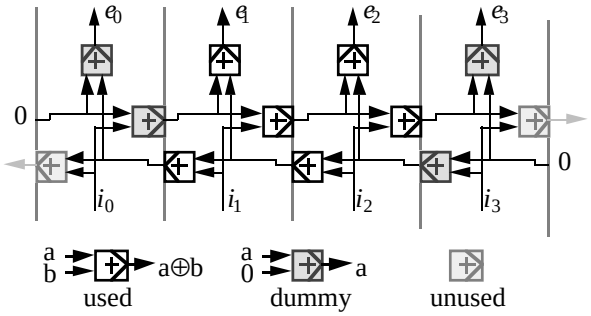

Figure 2: Block diagramm of a 4 bit parity check

If the Log-Likelihood Ratio (LLR) is used to represent the reliability of the bits, the output information e_2 of the binary operator is computed from the input information i_0 and i_1 by the relation [7]:

$$e_2 = i_0 \oplus i_1 = \ln\left(\frac{1+e^{i_0}.e^{i_1}}{e^{i_0}+e^{i_1}}\right) \qquad (4)$$

One can note, from figure 2, that the parity check processor can be easily dynamically reconfigurable. In that case irregular LDPC can be decoded as well with the proposed architecture.

2.5. Shuffle network

The shuffle network is realized with a q layer network. As shown in figure 3, each layer is progammed with a single bit s_i leading to a given permutation π_i if s_i =1 (π_i^1), or to the identity permutation (Id = π_i^0) otherwise.

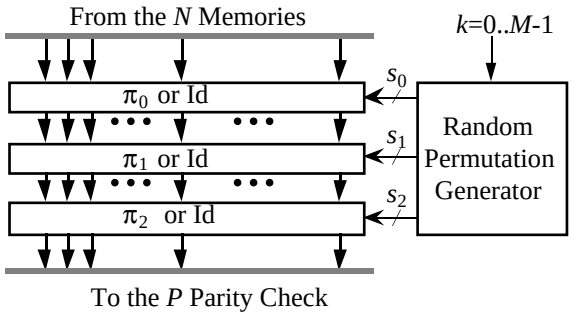

Figure 3: Interconnection network for q = 3.

A q bit word $s=(s_{q-1},...,s_1,s_0)_2$ defines simply the permutation π_s to be performed (see figure 2):

$$\pi_s = \pi_{q-1}{}^{s_{q-1}} \text{ o } ... \text{ o } \pi_1{}^{s_1} \text{ o } \pi_0{}^{s_0} \qquad (5)$$

The RPG is thus an hashing function from [0,M-1] to [0, 2^q-1] that associates to the iteration number k the value s=RPG(k), i.e. the index of the permutation.

For the unshuffling process, the network is exactly symmetrical.

2.6. Memory block

The management of the extrinsic information from iteration d to iteration d+1 is done directly in the memory block. The memory block is composed of 4 dual port RAM blocks, RAM I, Sa, Sb and E, as shown in figure 4.

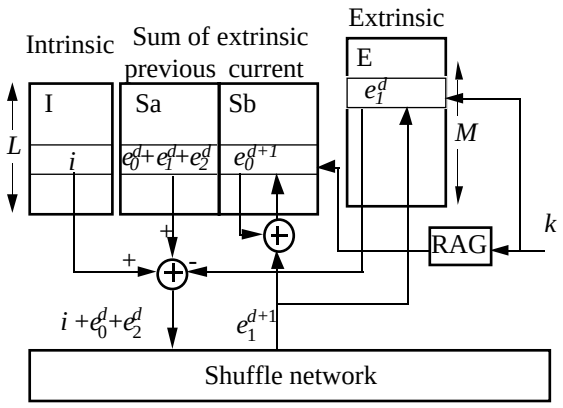

Figure 4: Management of extrinsic information in a memory bank

At clock cycle k of iteration d+1, the intrinsic information i of the current data (RAM I) is added to the sum of all the extrinsic information computed during iteration d (RAM Sa). Simultaneously, the extrinsic information generated by the current parity check during the previous iteration is retrieved from RAM E and subtracted from the total. Then, the result is sent to the parity check. The updated extrinsic information computed by the parity check is accumulated with the previous extrinsic information of the iteration d+1 (RAM Sb). This information replaces also the old extrinsic information in RAM E (note that the size of RAM E is equal to M). During the next decoding iteration, the roles of RAM Sa and RAM Sb are flipped and so on.

The use of a RAM size of 3L for I, (decomposed of 3 areas Ia, Ib and Ic, each of size L) allows to load, decode and output simultaneously 3 consecutive blocks. According to (2), during the DM decoding cycles of the a block (stored in area Ib, for example), the previous decoded block (respectively, the next block), can be output (can be loaded) at the same rate than the clock from Ic (to Ia). For the next decoding block, the role of the areas Ia, Ib and Ic are permuted. Thus, the decoding hardware can be fully used.

3. Example of realization

To illustrate the relation between architectural characteristics and the obtained LDPC code, a very simple example is first given. Then simulation results for a 4096 length, half rate LDPC code based on the proposed architecture are given.

3.1. A simple decoder'first code

Let us consider the code with the architecture parameters of table 1:

N	P	M	L	q	D	r	$\{c_i\}_{i=0..P-1}$
6	2	3	2	2	4	0.5	{3,3}

Table 1: Characteristic of the hardware.

Let us take the M=3 length sequence of the N=6 RAG: {0,1,1}; {1,1,0}; {1,0,1}; {0,0,1}; {1,0,1}; {1,1,0}. Let us assume that the M=3 length sequence of the RPG is {1,0,3} and finally, let us choose the q=2 permutations π_0 and π_1 of the shuffle network equal respectively to (3,5,0,1,2,4) and (4,0,5,2,3,1). The NL=12 data of the codeword are naturally indexed with their position in the memory bank: address i of MEM j contains data number $Lj + i = 2j + i$.

During the first cycle of a decoding iteration, data number 0, 3, 5, 6, 9, 11 are retrieved from the RAM, after the shuffling (π_0 , since RPG(0)=1), check nodes 0 and 1 receive respectively data {6,11,0} and data {3,5,9}. Table 2 shows the operation performed during the M=3 cycle of a decoding iteration.

	Read data	shuffle	Check 0	Check 1
0	{0,3,5,6,9,11}	π_0	{6,11,0}	{3,5,9}
1	{1,3,4,6,8,11}	Id	{1,3,4}	{6,8,11}
2	{1,2,5,7,9,10}	π_1 o π_0	{5,7,9}	{1,2,10}

Table 2: sequence of a decoding iteration.

The resulting factor graph of this code is given in figure 5.

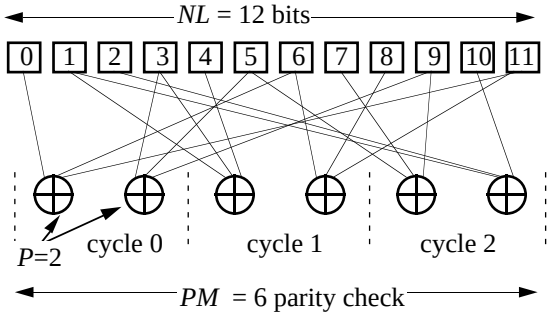

Figure 5: Bipartite graph obtained with the chosen RAGs and RPG.

Note that in the present example, the bit numbers 6 and 11 have 2 differents parity check nodes in common

(distance in the graph of 2). Generally, this type of configuration leads to a weak code. The generation of the RAG and RPG sequences is thus constrained in order to avoid such configuration.

3.2. SIMULATION RESULTS

Simulation of this architecture has been made with the parameters of table 3.

N	P	M	L	q	D	r	$\{c_i\}_{i=0..P-1}$
32	5	409	128	3	10	0.50	{6,6,6,7,7}

Table 3: Simulation parameters.

Figure 6 gives the performance obtained for different signal to noise ratios. Each point is obtained with the transmission of 250 blocks. The data are coded with one bit of sign, and 6 bits for the value (4.2, i.e. with two bits after the decimal point). The internal representation of the sums of extrinsic information is on 8 bits (6.2) but information is saturated to 6 bits before entering the shuffle network. The equation (4) is quantized and saturated: the 6 bit result of the function are read from a simple access in a 64x64 look-up table.

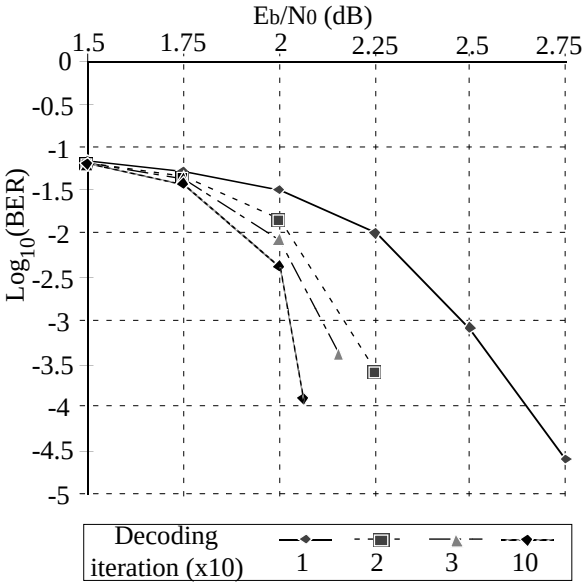

Figure 6: Simulation results of a LDPC (4096, 2045).

A simulation using the floating point representation of the LLR has also been performed. For a SNR of 1.75 dB, all of the 250 received blocks are decoded in less than 40 iterations. This result shows that the constructed code has similar performance than regular LDPC codes of same size. The one-chip decoder architecture proposed in the paper has performance, for a BER of 10^{-4}, similar to the TURBO4 chip [3], as shown in table 4.

Iterations	1	2	3	5	10
LDPC	2.65	2.3	2.2	-	2.03
TURBO4	3.5	2.4	1.9	1.6	-

Table 4: SNR (dB), needed to obtain a BER of 10^{-4}

A hardware complexity evaluation of the proposed architecture, as well as the definition of an efficient encoding process, have still to be done in order to make a full comparaison with the TURBO4 chip.

4. CONCLUSION

We believe that conceiving of the hardware architecture prior to, or at least jointly with, the choice of the code is a key that leads to efficient high speed hardware decoders for LDPCs and similar codes. The excellent performances (see table 4) obtained with the proposed high speed LDPC decoder architecture shows the efficiency of the "decoder first code design" approach.

Many topics have still to be addressed: estimation of the hardware complexity; development of an efficient coding strategy, extend the methodology to irregular LDPC or Tanner codes. Another promising direction of research is to use non uniform RAGs in order to construct codes with different levels of errors correction capability. The unequal error protection is very useful for some applications.

Acknowlegments

Thanks for Eric Perpere for the simulations on the LDPC code architecture, made during his training period at ENST (mai to august 2000).

REFERENCES

[1] R.G. Gallager, "Low-Density Parity-Check Codes", Cambridge, Massachusetts: M.I.T. Press, 1963.

[2] C. Berrou, A. Glavieux, P. Thitimajshima, "near Shannon limit error-correcting coding and decoding", in *Proceeding of ICC'93* (Geneve, Switzerland), pp. 1064-1070, May 93

[3] M. Jézéquel, C. Berrou, C. Douillard, P. Pénard "Caracteristics of a sixteen-state turbo-encoder/decoder (turbo4)", *Proceedings of the International Symposium on Turbo-Codes topics*, Sept. 97, Brest, France.

[4] T. Richardson, A. Shokrollahi, R. Urbanke, "Design of Provably Good Low-Density Parity Check Codes", submitted to *IEEE trans. on Information Theory*.

[5] D.J.C. MacKay, R.N. Neal, "Near Shannon limit perfomance of low density parity check codes", *Electronics Letters*, vol 33, no 6, March 1997, pp. 457-458.

[6] G. D. Forney, "On Iterative Decoding and the Two-Way Algorithm", *Proceedings of the International Symposium on Turbo-Codes topics*, Sept. 97, Brest, France.

[7] G. Battail, H.M.S El.Sherbini, "Coding for radio channels", *Ann. Télécommun.*, vol. 37, no 1-2, pp. 75-94, Jan./Feb. 1982.