Project	**IEEE 802.16 Broadband Wireless Access Working Group <http://ieee802.org/16>**
Title	**FEC proposal: use of Block Turbo Code (BTC) for IEEE 802.16.1 Air Interface Standard**
Date Submitted	**2000-07-10**
Source(s)	Christophe MARTIN ALCATEL 5, rue Noël Pons 92734 Nanterre Cedex, FRANCE Voice: +33 1 46 52 47 25 Fax: +33 1 46 52 35 93 mailto:christophe.martin@alcatel.fr
Re:	This is a response to a Call for Contribution for BWA FEC solutions (IEEE802.16.1p-00/06), dated 2000-05-04
Abstract	This contribution present a BWA FEC solution based on Block Turbo Code (BTC)
Purpose	Assist the IEEE 802.16.1 working group in evaluating proposed FEC system, for both mode A and mode B of the current PHY draft (IEEE802.16.1pc-00/29r1)
Notice	This document has been prepared to assist IEEE 802.16. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.
Release	The contributor grants a free, irrevocable license to the IEEE to incorporate text contained in this contribution, and any modifications thereof, in the creation of an IEEE Standards publication; to copyright in the IEEE's name any IEEE Standards publication even though it may include portions of this contribution; and at the IEEE's sole discretion to permit others to reproduce in whole or in part the resulting IEEE Standards publication. The contributor also acknowledges and accepts that this contribution may be made public by IEEE 802.16.
Patent Policy and Procedures	The contributor is familiar with the IEEE 802.16 Patent Policy and Procedures (Version 1.0) <http://ieee802.org/16/ipr/patents/policy.html>, including the statement "IEEE standards may include the known use of patent(s), including patent applications, if there is technical justification in the opinion of the standards-developing committee and provided the IEEE receives assurance from the patent holder that it will license applicants under reasonable terms and conditions for the purpose of implementing the standard." Early disclosure to the Working Group of patent information that might be relevant to the standard is essential to reduce the possibility for delays in the development process and increase the likelihood that the draft publication will be approved for publication. Please notify the Chair <mailto:r.b.marks@ieee.org> as early as possible, in written or electronic form, of any patents (granted or under application) that may cover technology that is under consideration by or has been approved by IEEE 802.16. The Chair will disclose this notification via the IEEE 802.16 web site <http://ieee802.org/16/ipr/patents/letters>.

FEC proposal: use of Block Turbo Code (BTC) for IEEE 802.16.1 Air Interface Standard

Christophe MARTIN, Alcatel, christophe.martin@alcatel.fr

1 Product Code Description

1.1 General principle

The concept of Product Code is a simple and relatively efficient method to construct powerful Codes using simple linear block Codes. It is introduced in [1] and [2]. A $N_C \times N_L$ matrix is used to encode $K_C \times K_L$ information symbols, bits here. A row code (N_L, K_L) and a column code (N_C, K_C) are then applied to the rows and the columns, $R_i = K_i/N_i$, $i = \{C,L\}$ is the corresponding coding rate. The Product Code is a (N,K) linear code, $N = N_C \times N_L$, $K = K_C \times K_L$, with a rate of $R = R_C \times R_L$. A codeword of the Product Code can be seen in 4 parts, the information part, the two single parity check parts and the double check part, as it can be seen in the Figure 1.

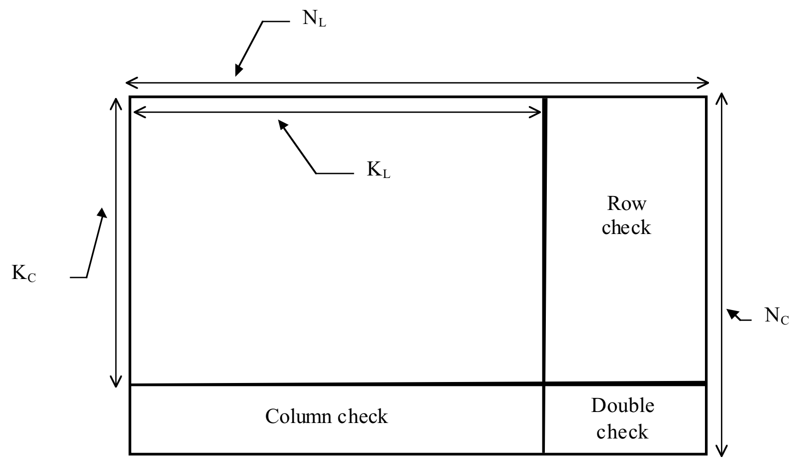

Figure 1 - Product Code codeword -

Two basic but interesting results whose demonstrations are not widely known can be stated here about the elementary Product Codes :

It is easy to show that the two single check parts, the row part and the column part, plus the double check part, are composed of codewords of the row code and the column code respectively. In the decoding, these codewords are decoded by the same decoders as for the information parts.

- The Hamming distance of the Product Code d, being equal to $d_C \times d_L$, will be maximum if the row code and the column code are identical. Preferentially, good Product Codes should have its row and column codes as similar as possible, identical if possible, for a big distance.

1.2 Shortening and extension of codes

It is possible to shorten a row or a column code by replacing information bits by known bits (generally equal to zero) which are not to be transmitted. Shortening a (N,K) linear code by p gives a ($N-p$, $K-p$) linear code with the corresponding rate $R_p=(K-p)/(N-p)$.

It is also possible to extend a row or a column code by adding a parity bit to the redundancy bits. Extending the previous (N-p, K-p) linear code gives a ($N+1$-p, K-p) linear code with the corresponding rate $R_{p,ext}=(K-p)/(N+1-p)$.

In the case of a product code using a row code (N_L, K_L), shortened by p_L and extended, and a column code (N_c, K_c) shortened by p_C, the corresponding coding scheme is represented in the Figure 2.

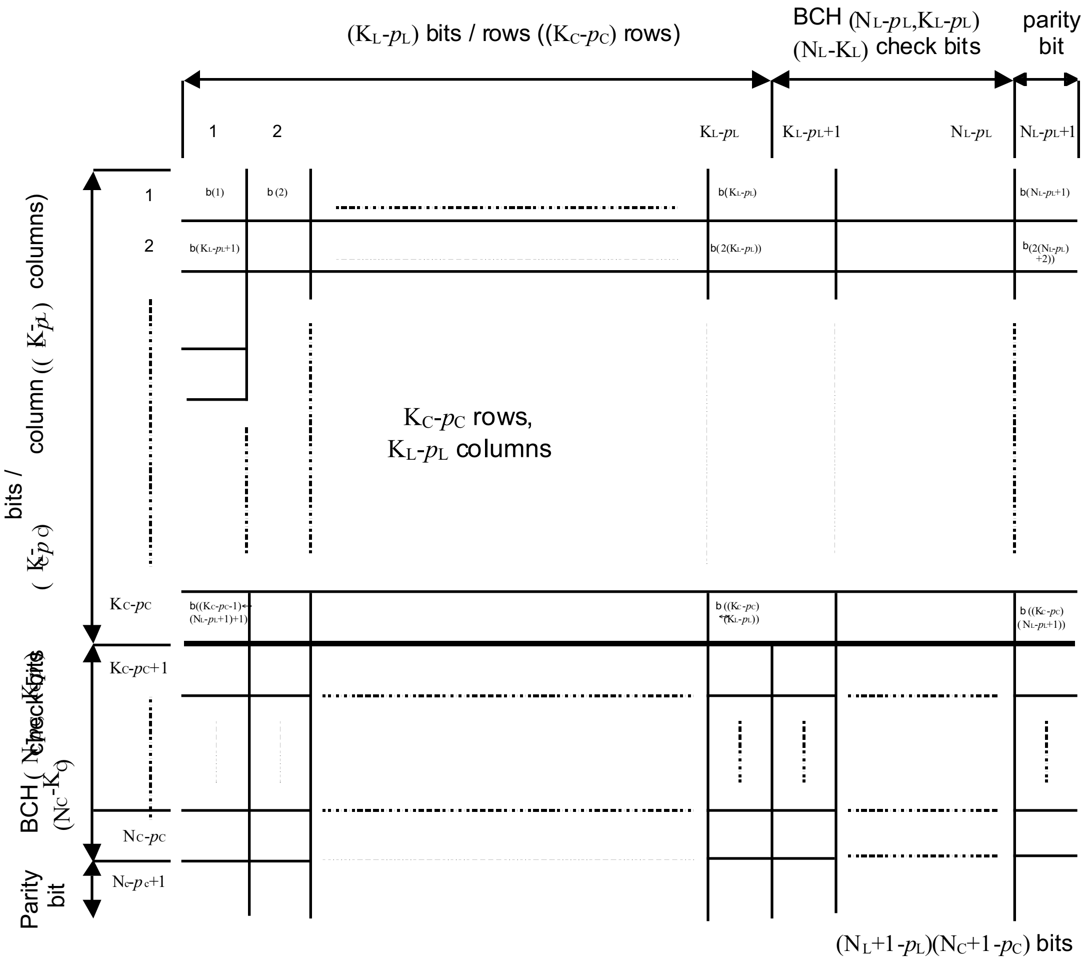

Figure 2 - Product code scheme -

The corresponding rate of this product code is $R=[(K_L-p_L)\times(K_c-p_C)]/[(N_L+1-p_L)\times(N_c-p_C)]$. The number of information bits is $(K_L-p_L)\times(K_c-p_C)$ and the number of coded bits, which is the number of bits to be transmitted, is $(N_L+1-p_L)\times(N_c-p_C)$.

Let be d_L and d_C the Hamming distance of the row and column codes before shortening and extension. The shortening does not change the Hamming distance of these codes and the extension of the row code increases its Hamming distance by one if it is odd. The Hamming distance of the product code is $d = (d_L+1)\times d_C$.

1.3 Use of stuffing bits

When the number of information bits is not equal to $(K_L-p_L)\times(K_c-p_C)$ (with $(K_L-p_L)\approx(K_c-p_C)$), extra stuffing bits may be used to encode the information part. They are known bits (generally equal to zero), interleaved with the information bits in order to construct a matrix to encode. After encoding, these stuffing bits are not transmitted. Every stuffing bit shortens the row code and the column code, which it belongs to. Practically, to keep codes with quite the same rate on all the rows, and on all the columns, stuffing bits are diagonally distributed in the information bit matrix, as detailed in Figure 3.

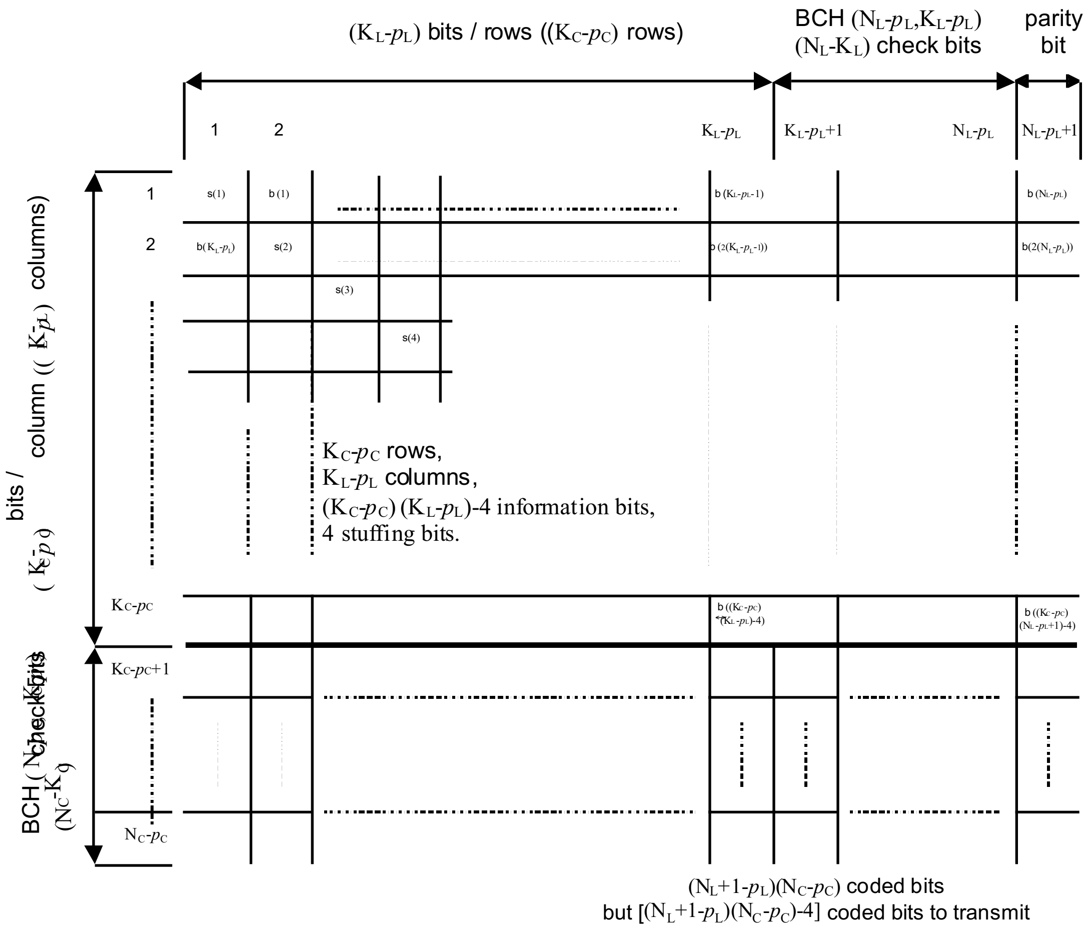

Figure 3 - Product code scheme with stuffing bits -

Let n_s be the number of stuffing bits (n_s=4 in Figure 3). The corresponding rate of this product code is R=[(K_L-p_L)×(K_C-p_C)-n_s]/[(N_L+1-p_L)×(N_C-p_C)-n_s]. The number of information bits is (K_L-p_L)×(K_C-p_C)-n_s, the number of coded bits is (N_L+1-p_L)×(N_C-p_C), and the number of bits to be transmitted is (N_L+1-p_L)×(N_C-p_C)-n_s.

The stuffing does not decrease the Hamming distance of the product code : $d \geq (d_L+1)\times d_C$.

1.4 Use of puncturing bits

Puncturing bits may be used in order to adjust the rate of a given product code. These bits (generally redundancy bits) are not transmitted. In reception, neutral bits (with a null reliability), are inserted at the right place in the received cell before decoding.

Every puncturing bit increases the rate of the row code and the column code it belongs to, but may shorten its Hamming distance. Practically, to keep codes with quite the same rate on all the rows, and on all the columns, puncturing bits are diagonally distributed in the encoded matrix, as detailed in Figure 4.

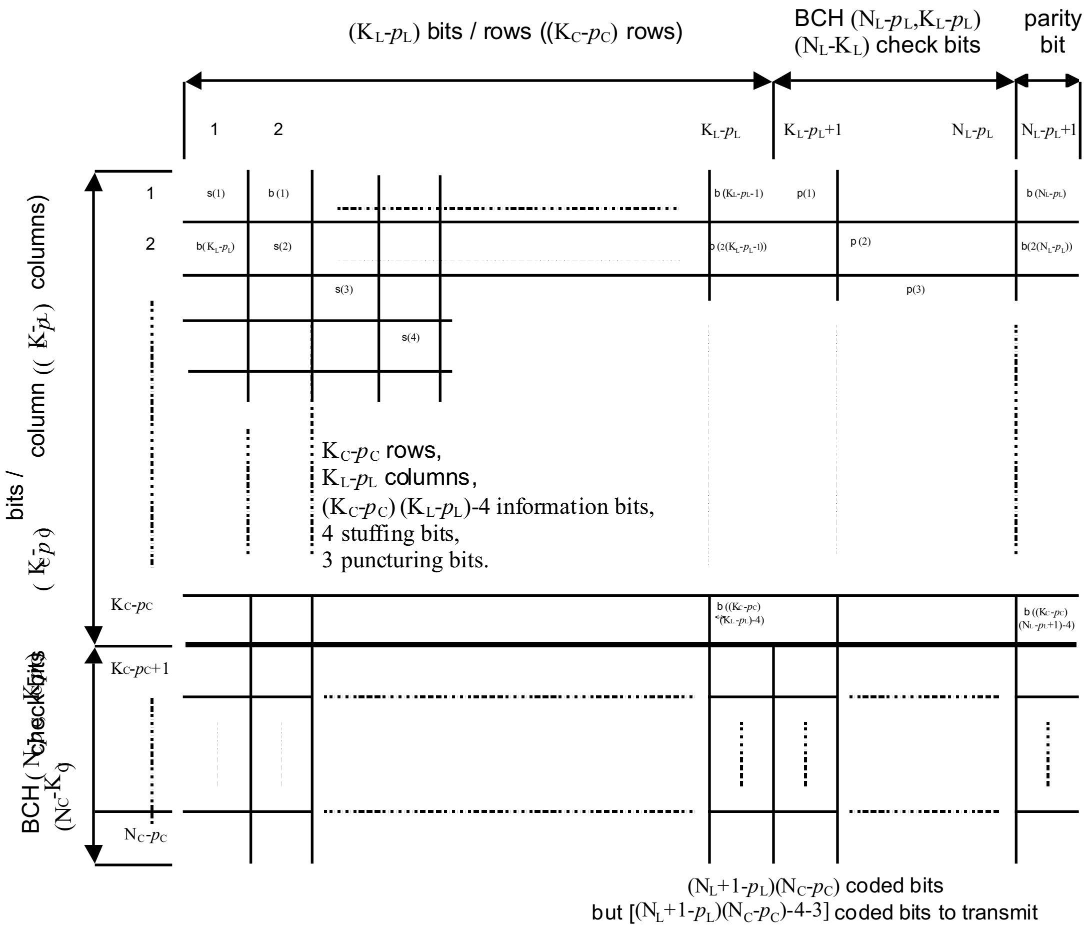

Figure 4 - Product code scheme with puncturing bits -

Let n_p be the number of puncturing bits (n_p=3 in Figure 4). The corresponding rate of this product code is $R=[(K_L-p_L)\times(K_c-p_c)-n_s]/[(N_L+1-p_L)\times(N_c-p_c)-n_s-n_p]$. The number of information bits is $(K_L-p_L)\times(K_c-p_c)-n_s$, the number of coded bits is $(N_L+1-p_L)\times(N_c-p_c)$, and the number of bits to be transmitted is $(N_L+1-p_L)\times(N_c-p_c)-n_s-n_p$.

The Hamming distance of the product code depends on the number of stuffed and punctured bits. As puncturing may decrease the code performance, it is not used in the proposed coding rate.

2 Decoder structure

2.1 Functional description

The decoding process is a Soft Input decoding. The inputs of the decoder are signed values received from the transmission channel, referred to as symbols. A row decoding and a column decoding are applied to the rows and the columns of the matrix, respectively. These row and column decoders are referred to as component decoders.

The proposed decoder is an iterative decoder. That is, the decoding process described above is repeated several times, three times in the proposed implementation (see Figure 5). This iterative structure influences the structure of the component decoders as follows :

- Each component decoder outputs soft values called *SoftOutput*, and not only a Hard decision sequence of bits.
- Each component decoder outputs soft values corresponding to the information symbols and the redundancy symbols.

- The component decoder output is a pondered value, depending on the information brought by the decoding (called *Extrinsic*) and the information brought by the channel (called *SoftInputID*), and defined by :

$SoftOutput = SoftInputID + \alpha \times Extrinsic$

Thus, *SoftInputID* and α are two complementary input of the decoder.

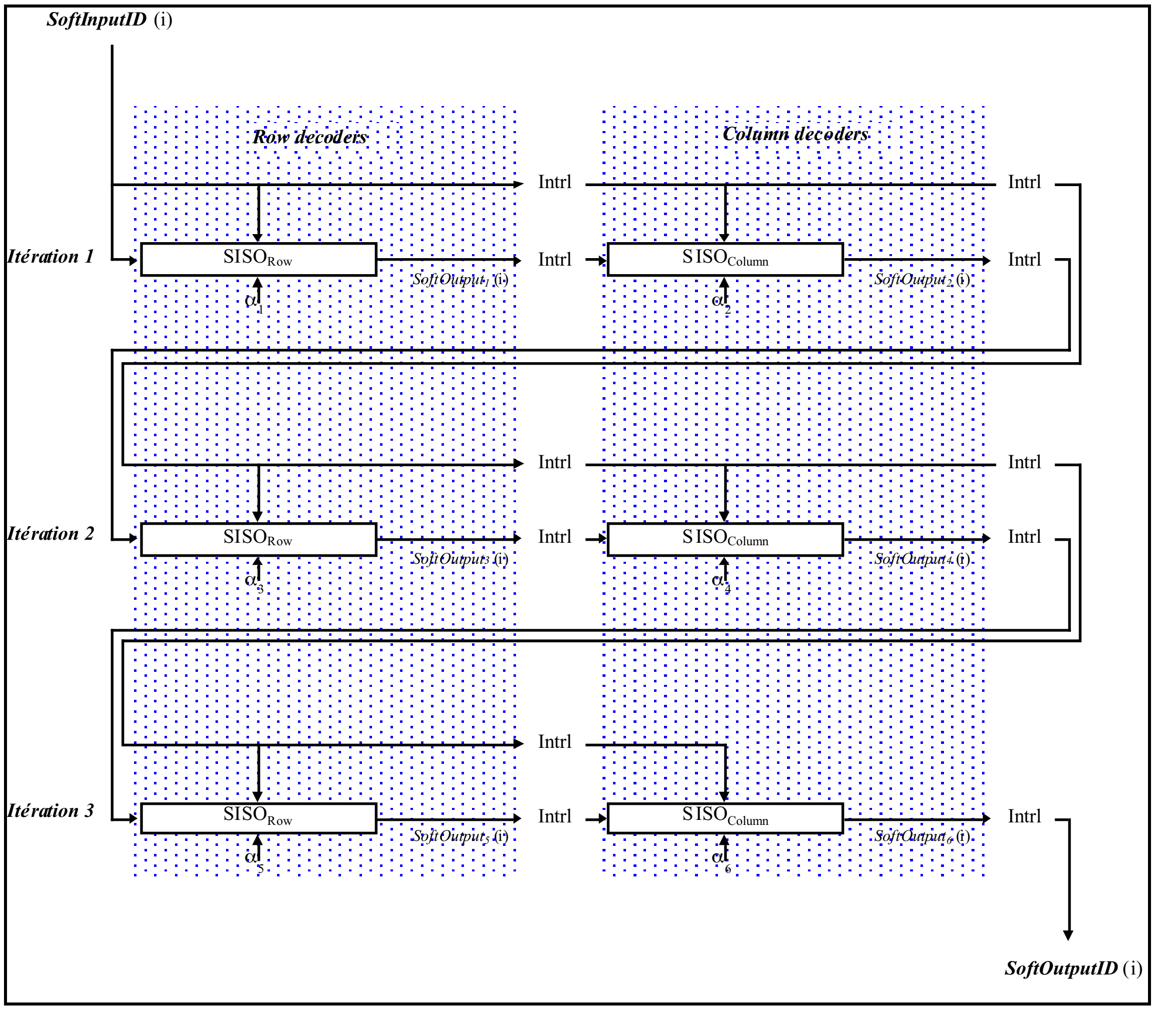

Figure 5 : Iterative decoder functional description (3 iterations)

The functional structure of the decoder is presented in the Figure 5. This architecture involves two kind of sub-functions :

- Interleaver. It consists in writing the incoming data in a memory, row by row (res. column by column), and then reading the memory column by column (res. row by row) at the output level.
- Component decoders. They are adapted either for decoding the rows code or for decoding the column code. Both row and column decoders have the same structure. Only their configuration parameters change.

The component decoders are Soft Input Soft Output decoders. Many algorithms shall be put into practice to compute the Soft Outputs. They should be divided into two families:

- Trellis based algorithm. Such algorithms are based on the trellis associated to the component block code.
- Augmented List Algorithms (ALD for Augmented List Decoding). Such algorithms produce a list of codewords near the received word to compute an approximation of the Soft Output.

For the simulations, an ALD component decoder is used, based on the Fang-Buda Algorithm (FBA). The decoder allows to decode any BCH code. A BCH code is characterized by the parameters (n,k,t) where n is the code word length, k is the uncoded information word length and t=(n-k)/2 is the correction capacity.

2.2 Implementation issues

The complexity of the decoder depends on the size of the component decoders (n and k) and the number of iterations.
A study of the functional complexity has been achieved for QPSK with the following code (1 PDU, rate _, 21/27x21/32 based on 26/32x21/32, i.e. t_1=1 and t_2=2), for QPSK modulation and for a clock frequency around 50 MHz. Data are quantified over 7 bits. For a complete iteration, that is to say for the 21/32 Soft Input Soft Output (SISO) decoding, the 26/32 SISO decoding and the corresponding interleavers, we get:

• SISO 21/32:	96 kgates,	2.2 kbit of RAM;
• SISO 26/32:	84 kgates,	2.2 kbit of RAM;
• Interleavers:	4 kgates,	32 kbit of RAM.

Thus the decoder complexity is about :
for one iteration, 1 PDU, rate _: 180 kgates and 4.5 kbytes of RAM.
This corresponds to the minimum required complexity.
When we use code working on 3 PDU ATM cells, codes are longer. Thus complexity increases. We computed the same kind of estimation for the following code (which is the most complex beyond the 3 codes with 3 PDU and the coding rate 1/2, 2/3 or 3/4): 3 PDU, rate 1/2, 36/49x36/49 based on 51/64x51/64. It gives:
for one iteration, 3 PDU, rate _: 365 kgates and 5.3 kbytes of RAM.

⇒ If we compare these estimated gate counts with the order of magnitude of the gate counts given in [3], we observe a wide difference. This is mainly due to the fact that the codes used here are BCH codes, with a correction capacity $t \leq 3$ in practice, whereas in the compared contribution, the authors only use parity codes and Hamming codes (t=0 ot t=1). Thus, in [3], algorithms are optimized for parity codes and Hamming codes, and they are less complex. On the other hand, we can easily achieve a coding rate 1/2 with a 424 bits PDU in 2 dimensions with a minimal distance d_{min}=24. Besides, it should be noticed that BTC decoders with complexity comparable to those presented in [3] can be designed with little degradation compared to those presented here (less than 0.5 dB, using SISO algorithms derived from the Chase algorithm)

If we want to use 16 QAM, we have two possibilities. The first one consists in working with 16 QAM symbols, in particular for the metric calculus. This should increase a little bit the complexity compare to QPSK. Another one consists in using multi-level coding (sometimes called pragmatic Treillis Coded Modulation), with for example BTC for the first level and a simple code for a second level, leading to a complexity a little bit increased also compared to QPSK, but this one is likely to have better results.

In reality, at a given clock frequency, in order to respect timing constraint, it may be necessary to pipe the treatments. This will induce an increase in the number of gates. For example, a design corresponding to a bad technology and poorly optimized libraries have been simulated. The target was an ASIC at 56 MHz. It was necessary to pipe some treatments, so that the decoding latency for 6 iterations was T=32 μs for 6 iterations. The results give then a very pessimistic estimation of the complexity, which could be reduced with good libraries and without piping some modules. We get:
Pessimistic ASIC design, 1PDU, rate 1/2: 880 kgates + RAM for 3 iterations (≈290 kgates/iteration)

In any case, double the clock frequency does not lead to double the gate number in the design. The clock frequency clearly is inversely proportional to the decoding delay and this delay changes also with chosen architecture (piped treatments) and cell size/coding rate. It should not be critical with BTC.
Concerning the flexibility, we can say that we can get real coding rates very close to target coding rate, using especially stuffing and code shortening and different code size. But, if we want to find coding rates with 1 ATM cell for example using the same base code (for example 21/32x26/32), we may not be able to find rates as close as

before, and it may be necessary to use puncturing. This implies that the performance may decrease compare to another code with the same coding rate but without puncturing.

The following table is a summary of the required BTC characteristics:

FEC scheme	BTC
Encoder complexity QPSK / 16 QAM / 64 QAM	15 kgates (based on AHA figures)
Decoder Complexity (QPSK) 1 and 3 ATM cell 1 iteration, rate 1/2	180 - 300 kgates for 1 PDU; >365 kgates for 3 PDU
Max. Frequency Clock (16QAM)	80 ~ 160 Mhz (based on AHA figures)
Overall Processing delay (1 and 3 ATM cell)	32 μs for 6 iterations in 56 MHz for 1 ATM cell;
Time to market impact (2002)	Feasible for 2002 (VHDL available now)
Error detection capability	Need to include CRC
Flexibility	Yes

Table 1 : BTC complexity

2.3 Error detection capabilitiy

As mentioned in [3] and [4], if needed, it is possible to achieve a frame error detection., either using an additional half-iteration, or using extra bits for a CRC mechanism. It is worth noting that these CRC bits may be the stuffing bits: in this case, the stuffing bits would be transmitted as CRC bits. Some results can be found in [4] for CRC dimensioning.

3 Simulation results

Simulations are carried out with QPSK modulation and 6 iterations. A PDU size is 424 bits, 53 bytes. Theoretical results are only approximated, for a reason of simplicity and because they are sufficient to give a good idea of the performance. In fact, for the calculus, we only use an approximation of the number of words of weight d_{min} for each code, instead of computing the complete spectrum of the code.
The following figure shows the theoretical results and the simulations results, for the Bit Error Rate (BER) and the Cell Error Rate (CER, the number of corrupted information blocks).

3.1 Packet size: 1 PDU, 424 bits

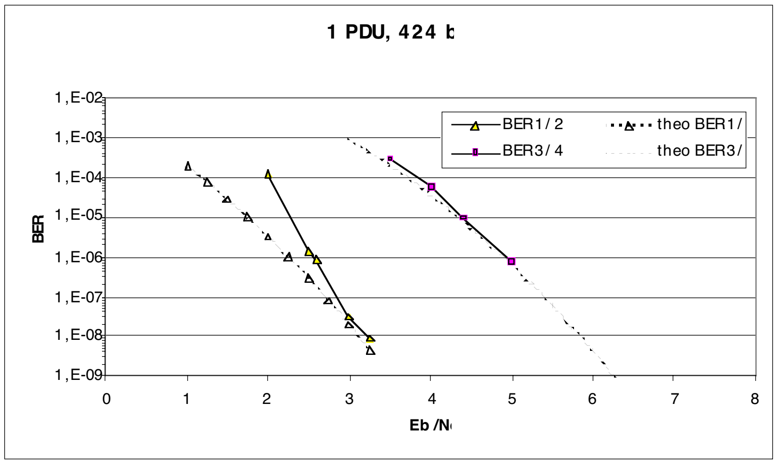

Figure 6 : BER performance for 1 PDU packet size

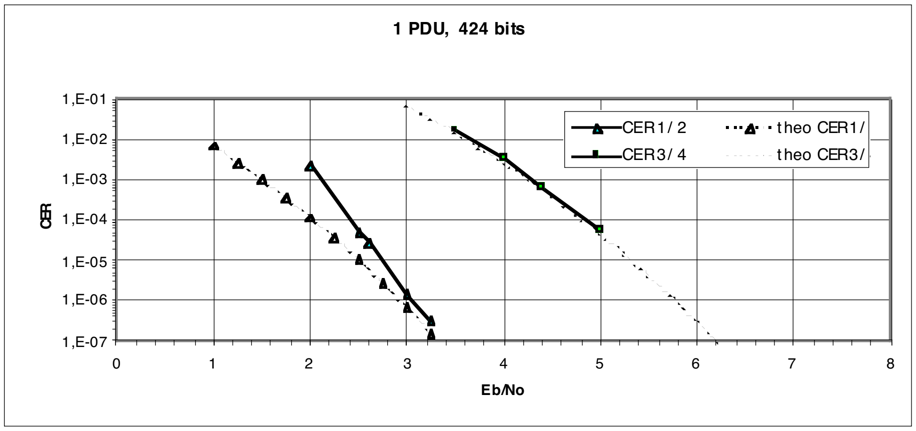

Figure 7 : CER performance for 1 PDU packet size

3.2 Packet size: 2 PDU, 848 bits

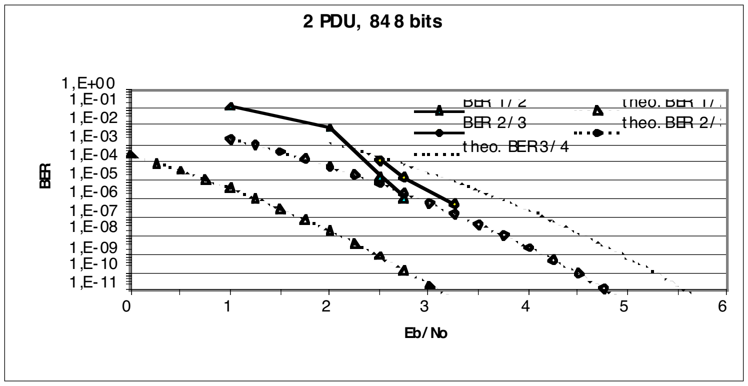

Figure 8 : BER performance for 2 PDU packet size

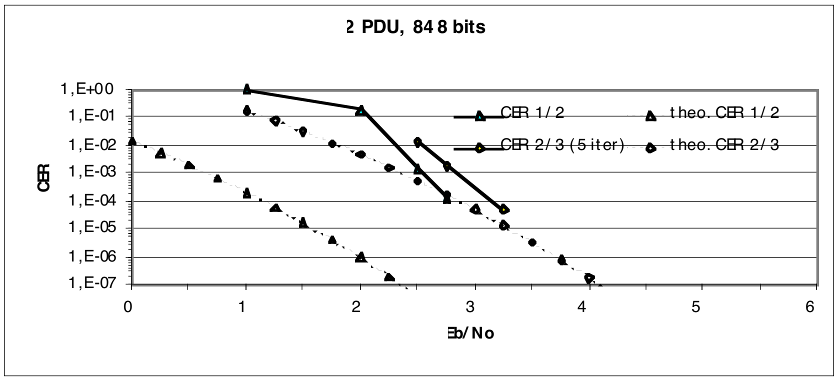

Figure 9 : CER performance for 2 PDU packet size

3.3 Packet size: 3 PDU, 1272 bits

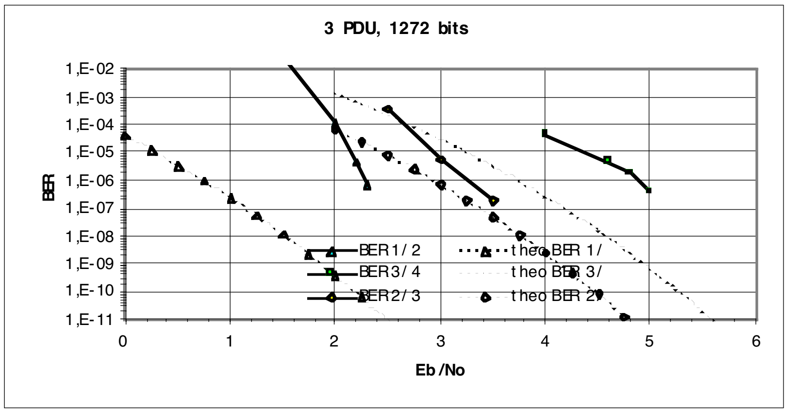

Figure 10 : BER performance for 3 PDU packet size

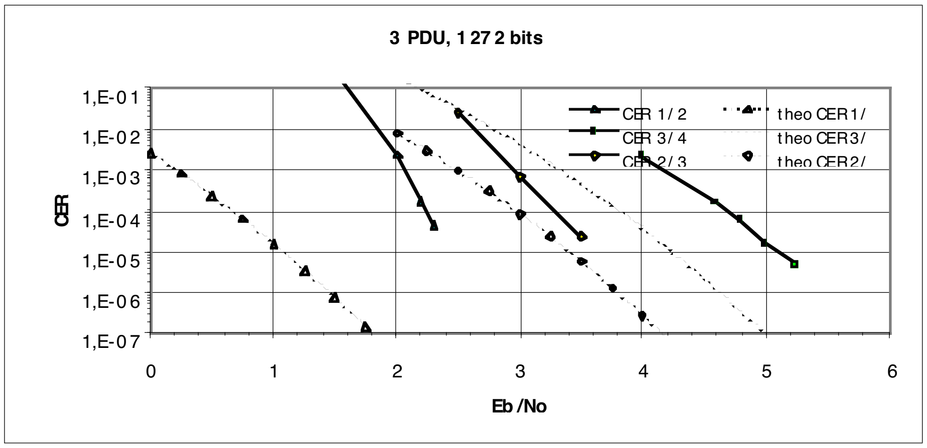

Figure 11 : CER performance for 3 PDU packet size

3.4 Summary table

In Table 2 and Table 3, we give the performance for different configurations. For each configuration, we find:

- a description of the code;
- the coding rate;
- for two BER values (10^{-6} and 10^{-11}), the E_b/N_0 and C/N values to get this BER, and the corresponding CER.

QPSK	424 bits, rate 1/2	424 bits, rate 3/4	848 bits, rate 1/2	848 bits, rate 2/3
Description	21/27x21/32 16 stuffing bits d_{min}=24	24/30x12/13 8 stuffing bits d_{min}=8	25/32x34/53 2 stuffing bits d_{min}=32	34/41x25/31 1+1 stuffing (*) d_{min}=16
Uplink (UL), Downlink (DL), or both (UL+DL)	UL+DL	UL+DL	UL+DL	UL+DL
Coded bits	848	562	1694	1270
Effective coding rate	0.5	0.754	0.501	0.668
Eb/No for BER=10^{-6} (simulated)	2.6	5	2.75	3.25
C/N for BER=10^{-6} (simulated)	2.6	6.8	2.75	4.5
Corresponding CER	2.8 10^{-5}	5.4 10^{-5}	10^{-4}	1.3 10^{-4}
Eb/No for BER=10^{-11} (theory, union bound))	4.15	7	3.25	4.8
C/N for BER=10^{-11} (theory)	4.15	8.8	3.25	6
Corresponding CER	3.6 10^{-9}	4.8 10^{-10}	1.2 10^{-9}	7 10^{-10}

(*) stuffing bits "n+p" means that (n+p) bit are used as stuffing bits, p are transmitted (in order to get an even number of bits), n are not transmitted.

Table 2 : Configuration for 424 bits and 848 bits

QPSK	1272 bits, rate 1/2	1272 bits, rate 2/3	1272 bits, rate 3/4
Description	36/49x36/49 23+1 stuffing bits (*) d_{min}=36	23/30x56/63 16 stuffing bits d_{min}=16	50/63x28/29 15+1 stuffing bits d_{min}=12
Uplink (UL), Downlink (DL), or both (UL+DL)	UL+DL	UL+DL	UL+DL
Coded bits	2378	1874	1696
Effective coding rate	0.535	0.679	0.75
Eb/No for BER=10^{-6} (simulated)	2.3 (1)	3.25	4.8
C/N for BER=10^{-6} (simulated)	2.6	4.6	6.6
Corresponding CER	4.2 10^{-5}	1 10^{-4}	6 10^{-5}
Eb/No for BER=10^{-11} (theory, union bound))	2.5	4.75	5.5
C/N for BER=10^{-11} (theory)	2.8	6.1	7.4
Corresponding CER	5.5 10^{-10}	1.4 10^{-9}	1 10^{-9}

(*) stuffing bits "n+p" means that (n+p) bit are used as stuffing bits, p are transmitted (in order to get an even number of bits), n are not transmitted.

(1) quite far from the theory (difference $\Delta \approx 1.5$ dB); thus the value for a 10^{-11} BER may be reach with C/N higher than the theoretical value (2.8 dB)

Table 3 : Configuration for 1272 bits

4 Conclusion

The Bloc Turbo Code is a coding scheme that presents:

- A very good coding gain, very useful for C/I problems encountered during the deployment;

- Flexibility in the coding rate;
- A good complexity / performance tradeoff;
- The opportunity to include a CRC;

Besides, some hardware implementing turbo codes already exist (see www.aha.com, mail to: turboconcept@enst-bretagne.fr)

It is thus suitable for both mode A and mode B IEEE 802.16.1 Air Interface Standard, for both uplink and downlink. We propose to use block turbo-code at least as an option.

5 References

[1] Claude Berrou, Alain Glavieux and Punya Thitimajshima, "*Near Shannon Limit Error Correcting Coding and Decoding: Turbo-Codes*", IEEE Int. Conf. On Comm. ICC'93, vol 2/3, May 1993, pp 1064-1071.

[2] A. Berthet, J. Fang, F. Buda, E. Lemois, P. Tortelier, " A Comparison of SISO Algorithms for Iterative Decoding of Product Codes", in ISCOM 99

[3] IEEE 802.16.1pc-00/32r1, "FEC proposal: Evaluation of Block-Turbo Codes and Higher Order Modulation for IEEE802.16.1 Air Interface Standard".

[4] IEEE 802.16.1pc-00/35, "Turbo-Product Code FEC Contribution"