Real Time Turbo Decoding of BCH Product Code on the DSP Texas TMS320C6201

André GOALIC(*) and Nadine CHAPALAIN(**)
(*)ENST de Bretagne, Technopôle Brest-Iroise BP-832, 29285 BREST, BRITTANY
(**)Mitsubishi Electric ITE, Im. Germanium 80, av. des Buttes de Coësmes 35700 RENNES
E-mail : andre.goalic@enst-bretagne.fr

Abstract – *To speed up and enhance real time DSP application developments, a new kind of tool called "Code Composer Studio (CCS)" can be used, in the Texas Instrument DSP environment. It allows an easy check of real time "Block Turbo Code (BTC)" behaviour. The studio also includes the new DSP analysis technology, the Real-Time Data Exchange (RTDX). Thus the encoder implemented on the host computer exchanges data with the decoder implemented on the DSP board via the RTDX link. Once sent back to the host the decoded data allows analysis and curves drawing. This paper also presents our first results concerning the implementation of the "Block Turbo Code (BTC) " on the Texas Instruments TMS320C6201 (1600 MIPS) fixed point DSP. After four iterations our implementation can reach – 400 kbits/s – in C and – 550 kbits/s – in Assembly language.*

Keywords : Block-turbo-code, TMS320C6201, real-time-processing, RTDX.

I.INTRODUCTION

Channel coding is one of the main part of a digital communication system. The use of redundancy bits helps the decoder to restitute the emitted sequence. Flexibility and adaptability are easily obtained by an implementation on a DSP chip.

While using product codes based for example on a BCH(32,26,4) scheme, the coding gain at 10^{-5} Bit Error Rate slightly exceeds 6 dB[1]. Before the coding process, the information bits are stored in a matrix. The BTC coder then, adds redundancy bits to rows and columns, the sets of rows and columns remaining codewords. An iterative process is used to decode the product code. Section II describes the decoding algorithm. Section III presents the demonstration platform using the Code Composer Studio (CCS) and the RTDX link. Then we will approach, in section IV, the problems of real-time decoder implementation. Section V will show some promising results, obtained using C and Assembly language. Finally, conclusion and future perspectives of the real time TCB implementation on a DSP chip are given.

II. SOFT DECODING OF BTC

II.1. The product code

Let us consider a linear block code C having parameters (n, k, δ) where n, k, δ stand for code word length, number of information bits and minimum Hamming distance respectively. The code rate is defined by the ratio r = k/n. The product code $P = C \otimes C$ is then obtained by :

- placing (k × k) information bits in an array of k rows and k columns.
- coding k rows and n columns using code *C*.

The parameters of the resulting product code *P* are given by : $N = n \times n$, $K = k \times k$, $\Delta = \delta \times \delta$, and the code rate : $R = r \times r$. The (n-k) last rows and the (n-k) last columns also are code words of *C.* A soft decision algorithm derived from the theoretical Log-Likelihood-Ratio (LLR), proposed by R. Pyndiah in 1993 is used to compute the extrinsic information.

II.2. The soft decision algorithm

Let $R = (r_1,\ldots,r_i,\ldots,r_n)$ be the output of the coherent demodulator, $E = (e_1,\ldots,e_i,\ldots,e_n)$ the transmitted code word and $N = (n_1,\ldots,n_i,\ldots,n_n)$ a noisy Additive White Gaussian vector of standard deviation σ. Then the receive vector $R = E + N$. The decoded optimum sequence is given by :

$$D = C^i \text{ if } \Pr\{ E = C^i | R \} > \Pr\{ E = C^j | R \}, \forall j \neq I \quad (1)$$

where $C^i = (c_1^i,\ldots, c_l^i, \ldots, c_n^i)$ is the i^{th} *C* code word and $D = (d_1,\ldots,d_l,\ldots,d_n)$ the decision. For noisy samples, the decoding rule is simplified into :

$$D = C^i \text{ if } |R - C^i|^2 < |R - C^j|^2, \forall j \neq i \quad (2)$$

The optimum decoding process includes important computation time, incompatible with high bit rate real time implementations. The Chase algorithm allows to reduce the computational time with small performance degradation, it only looks after the code word at Hamming distance within a sphere of radius (δ - 1), centred on $Y = (y_1,\ldots,y_l,\ldots,y_n)$ the hard decoded vector of *R*. To further reduce the number of reviewed code words, only the most probable code words within the sphere are selected by using channel information. This search procedure can be decomposed in three steps :

step 1 : determine the position of the 3 least reliable binary symbols of *Y* using *R*.

step 2 : form 8 test sequences T^q defined as all the combination of sequences with "0" or "1" in the m least reliable positions.

step 3 : decode $Z^q = Y \oplus T^q$ using an algebraic decoder. It determines 8 code words C^q, one of them, the candidate will be chosen by soft decision and a second one, the concurrent, will be used for weighting.

III. CODE COMPOSER STUDIO-RTDX LINK

Designed for the TI high performance DSP TMS320C6xxx, Code Composer Studio tightly integrates the capabilities of :
- real-time data exchange between host and target
- real-time analysis and data visualization.

The demonstration platform (fig. 1) includes two parts :
- the first part is implemented in the host computer, generation of the matrix of information bits, coding rows and columns and sending the noisy samples to the decoder via the RTDX link. This part also includes BER computation before drawing curves.
- the second part is implemented in the DSP, it is the decoding process. Each noisy frame is decoded before sending back the matrix to the host via the RTDX link, then statistics can be made.

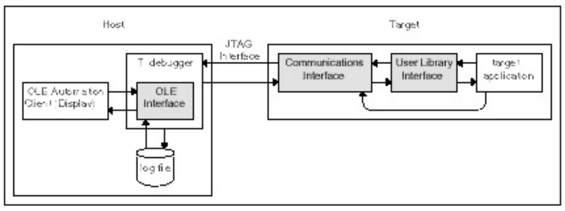

fig. 1 : Demonstration platform

VI. COMPUTATION

The Turbo Decoder (TD) is fed by a block of 1024 samples quantizied on 8 bits. The TD itself includes four main tasks, each one having to be processed on each row and column at each iteration :
- computation of decision vector *Y* using the received samples *R* and checking its parity, looking for and ordering the m least reliable binary symbols.
- computation of vector Z^q syndromes using an algebraic decoder. This decoder uses a precomputed double input table dedicated to the BCH(32,26,4) code. According to vectors Z^q bit values, the decoder checks the wrong bit positions, if the input vector is not a code word. After possible correction C^q belongs to the code words set.
- computation of the Euclidian distances between code words { C^q } and vector *R*. The code word with the maximum value becomes the selected one and the nearest will become the “concurrent”. In this real time implementation, we only compute one “concurrent”.
- computation of the extrinsic information bits.

IV.1 Syndrome computation

The algebraic decoder uses the double inputs table *SYND*[*S*][*j*] to check the possible wrong bit for the BCH(32,26,4) code. The words Z^q are 31 bit length words, $z_j{}^q$ is the value of the jth bit of the word Z^q. Let us recall the wrong bit checking procedure :

for j = 0,..,30 if $z_j{}^q = 1$ then *S* = *SYND*[*S*][*j*].

The last value of *S* indicates the wrong bit position if *S* is less than 31. The procedure has to be repeated for each word Z^q, $q = 0, \dots, 2^m - 1$. Ordered according to their position in the word (I_0, I_1, I_2) (fig. 2), they allow an efficient and fast method to compute in a single loop the syndromes.

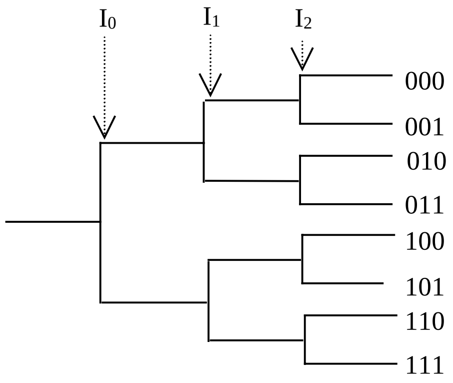

fig. 2 : Syndromes tree scheme

The new procedure implemented in real time environment can be summarised as follows (y_j is the value of the i^{th} bit of the word *Y*) :
- for j = 0,.., I_0-1 if $y_j = 1$ then *S* = *SYND*[*S*][*j*]

of course, this value S is the same for the all the words Z^q.
- j = I_0 : if $y_j = 1$ then $S_0 = SYND[S][j]$, $S_1 = S$
else $S_1 = SYND[S][j]$, $S_0 = S$

S_1 is the updated value of the set $\{Z^q\}_{q=0,\dots,3}$ and S_0 the updated value of the set $\{Z^q\}_{q=4,\dots,7}$.
- for j = I_0 + 1,., I_1-1if y_j = 1 then $S_1 = SYND[S_1][j]$,

$S_0 = SYND[S_0][j]$.

At I_1, I_2 bit positions, nodes are added to the tree allowing computation of S_{111}, S_{110}, S_{101}, S_{100}, S_{011}, S_{010}, S_{001}, S_{000}. The set $\{S_{lmn}\}_{l,m,n\in\{0,1\}}$ corresponds to the wrong bit positions into code words $\{C^q\}_{q=0,\dots,7}$, if $S_{lmn} < 31$.

IV.2. Euclidian distance

To update the selected code word C^d bits reliability, the decoder needs to compute the extrinsic information after each elementary decoding process. The Chase algorithm generates 8 code words C^q. The distance between vector R and code word C^q can be expressed as follows :

$$M^q = \left|R - C^q\right|^2 = \sum_{l=1}^{n}\left(r_l^2 - \left(c_l^q\right)^2\right) - 2\sum_{l=1}^{n} r_l c_l^q \qquad (4)$$

For all code words C^q, the first term has the same value. Code word C^d at minimum distance from R, also sets at its maximum value the term $\sum_{l=1}^{n} r_l c_l^q$.
The code words $\{C^q\}$only have a few different bits. If we consider the set of bits B, including the m least reliable bits and the set of bits corresponding to the syndromes : B = { I_0 , I_1 , I_2, $\{S_{lmn}\}_{l,m,n\in\{0,1\}}$}, the following term $\sum_{l=1, l\notin B}^{n} r_l c_l^q$ takes the same value for each code word of the set $\{C^q\}$. Thus, in a real time environment it has not to be computed. The real time algorithm perform the following steps :
- using syndrome computation strategy, we begin to calculate the m least reliable bits contributions to the sum for each code word according to chosen value.
- for each code word we compute the contribution corresponding to the syndrome.
The code word at minimum Euclidian distance from R is then chosen as C^d. We also look after one “concurrent” C^c to update the bits soft decision.

IV.3. Soft decision

Once the code word C^d found, the decoder only looks after one “concurrent” code word C^c, at the distance minimum from C^d. The value r_j’ can be expressed as follows :

$$r'_j = \frac{1}{2}\left(\sum_{l=1}^{n} r_l c_l^d - \sum_{l=1}^{n} r_l c_l^c\right) c_j^d \qquad (5)$$

then for all bits such as $c_j^d \neq c_j^c$ the soft decision of the bit r_j takes the following value :

$$w_j = \frac{1}{2}\left(\sum_{l=1}^{n} r_l c_l^d - \sum_{l=1}^{n} r_l c_l^c\right) c_j^d - r_j \qquad (6)$$

otherwise, for $c_j^d = c_j^c$, $w_j = \beta c_j^d$ where β is a constant function of the BER and optimized by simulation.

On receiving the matrix [R] corresponding to a transmitted code word [E] of the product code, the decoder decodes the rows (or columns) of the matrix, estimates the LLR and gives the output [W]. The decoding procedure described above is then iterated. We can illustrate the procedure as follow (fig 3) .

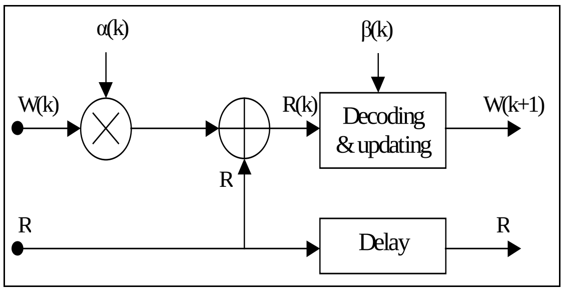

fig 3 : principle of an iteration

k: the iteration number, W: the extrinsic information, α: the weighting parameter, β: the parameter function of the BER, optimized by simulation. R(k) is expressed as follows : R(k) = R + α(k)W(k).

Extrinsic information updating depends on the two code words, C^d and C^c, found at step 3 :

$$w_j = \begin{cases} \sum_{l=1, l\neq j}^{n} r_l c_l^d p_l & \text{if } c_j^d \neq c_j^c \\ \beta \times c_j^d & \text{else} \end{cases} \qquad (7)$$

$$\text{where : } p_l = \begin{cases} 0 \text{ if } c_l^d = c_l^c \\ 1 \text{ if } c_l^d \neq c_l^c \end{cases} \qquad (8)$$

V. IMPLEMENTATION

V.1 Development and Test Environment

Hardware and software environment, used for the Turbo Decoder implementation consist of the evaluation module TMS320C6201 EVM provided by Texas Instruments. The software environment includes all the tools required for benchmarking. Two series of test have been conducted on the station, the first in C and the second in Assembly language.

V.2 C language experimentation

The C language program under test only includes the four tasks stated before. It is optimised with the help of the TI C Compiler that takes advantage of the CPU architectural features. All parameters are quantizied on 8 bits (char), 16 bits (short) and 32 bits (int).

One way to reduce the number of instructions is to keep when it's possible the parameters in the CPU. Thus, local parameters can stay in the CPU registers decreasing the number of access memory. In this implementation, the variables that are used during the four tasks are declared as global variables and the ones that are needed only for one task are defined as local variables. Thus, the number of frame reaches 582 per second, that is to say close to 400 Kbits/s.

V.2 Assembly language experimentation

To improve decoder performances the C functions are rewritten in Assembly language, the DSP environment remaining the same. For our application all the C functions have been rewritten in Assembly language.

Table 2 shows the decrease of the number of cycles when using Assembly language.

The new bit rate in assembly language is close to 550 Kbits/s. It appears clearly that the first task, which includes sort operations, requires soft pipelining to be optimized.

VI. CONCLUSIONS-PERSPECTIVES

This paper has presented the implementation of a Turbo decoder in C and Assembly language, on the DSP TMS320C6201 (1600 Mips version). Regarding the bit rate reached after 4 iterations – 400 kbits/s (C language) and 550 kbits/s (Assembly) –, those results are very promising and shows that acceptable performance can be achieved with C code without the development effort required with hand-written assembly code. However the Assembly language allows to use efficiently the computation power. This first version makes an important use of parallelism but poorly exploit the high capacity of the software pipelining. The portability of the C code on new TI chips will allow to highly increase the bit rates with only a little development investment.

TASKS	Number of cycles (functions)	
	C language	Assembly
Updating Input Hard decoding		427
Hard decoding	324	
Syndrome computation	220-324	212
Euclidian distance Candidate selection	220 96	134
Soft decision	260	135
Main task		60
Main task Updating Input	140	
Total	1338	968

Table 2 : Comparison C and Assembly language

Acknowledgement : The authors would like to thank TEXAS INSTRUMENT European 'ELITE Program' to have made this project possible while providing the DSP development environment

REFERENCES

[1] A. GOALIC, R. PYNDIAH, "Real-time Turbo-decoding of product codes on a digital signal processor", GLOBECOM '97 Phoenix pp. 624-628 V.2.

[2] C. BERROU, A. GLAVIEUX, P. THITIMAJSHIMA, "Near Shannon limit error-correcting coding decoding : Turbo Codes", IEEE Int. Conf. On Commun. ICC'93, vol. 2, May 1993, pp. 1064-1071.

[3] R. PYNDIAH, A. GLAVIEUX, A. PICART and S. JACQ, "Near optimum product codes", Globecom'94 , San Francisco, Nov. ,Dec. 1994 , pp. 339-343.

[4] Code Composer Studio, Texas Instruments.

[5] Real-Time Data Exchange, Texas Instruments.