Universitatea “Politehnica” din Bucuresti
Facultatea de Electronică, Telecomunicaţii și Tehnologia Informaţiei

Simularea software a codurilor LDPC și analiza performanţelor

Proiect de diplomă

Prezentat că cerinţă parţială pentru obţinerea titlului de Inginer în domeniul domeniul Electronică și Telecomunicații

programul de studii de licenţă Electronică Aplicată și Tehnologia Informației

Conducători stiinţifici

Conf. Dr. Ing. Ştefan STĂNCESCU
As. Univ. Drd. Cătălin SANDU

Absolvent

Iulian Cosmin FLORESCU

2014

CUPRINS

Capitolul I. Introducere

Teoria codurilor este prezentă în viața noastră mai mult decat am crede, pornind de la simpla utilizare zilnică a telefonului mobil, pană la securitatea și integritatea datelor de interes national. Cateva exemple de utilizare a teoriei codurilor:

- Protecția informației prin transmiterea ei repetată și decizie majoritară
- Protecția informației digitale prin bitul de verificare al parității.
- Protecția inregistrarii informației pe suport magnetic prin coduri ciclice corectoare de erori simple sau muliple
- Protecția vorbirii digitale comprimate în comunicațiile mobile (GSM) prin coduri ciclice detectoare de erori combinate cu coduri convolutionale corectoare de erori: în sistemul GSM codarea vorbirii se realizează utilizand o varianta imbunătățită de sistem de compresie bazat pe metoda predicției liniare în care fiecare cadru de vorbire având durata de 20ms este reprezentat prin 260 biți. După cum se vede în figură, biții sunt impartiti în trei clase: 50 biți foarte importanți pentru calitatea vorbirii, 132 biți importanți și 78 biți nu foarte importanți.
- Protecția informației în comunicatii sătelitare și misiuni spatiale utilizand variante de coduri ciclice nebinare, corectoare de erori multiple și/sau de erori în pachete că de exemplu coduri Fire, coduri Bose-Chauduri-Hockenguem (BCH), coduri Reed-Solomon. [3]

Claude Shannon publică în 1948 un document care avea practic să creeze domeniul teoriei informației și care avea să stabilească directiile de cercetare în acest domeniu pentru urmatoarea jumatate de secol[referinta shannon]. După ce mult timp viteza de trasmisiune pe liniile telefonice a fost blocata la 9.6 kilobiți pe secundă lucrarea lui C.E. Shannon avea să raspunda unor întrebari fundamentale pentru teoria transmisiunii: Exista coduri care pot mari viteza de trasmisie a datelor? Daca da, cât de mult? Și care sunt aceste coduri?

Shannon spune în lucrarea să că orice canal de transmisii de date - o linie telefonica, o bandă radio, un cablu de fibra optică, ar putea fi caracaterizat prin doi factori: lăţimea de bandă și zgomotul. Lăţimea de bandă este gama de frecvenţe electronice, optice sau electromagnetice, care pot fi folosite pentru a transmite un semnal; zgomotul este orice poate perturba semnalul. Luând în considerare un canal cu anumite caracteristici ale lăţimii de bandă și ale zgomotului, Shannon a arătat cum să se calculeze rata maximă la care datele pot fi trimise prin canale cu o eroare zero. El a numit acea rată capacitatea canalului, dar astăzi este adesea numită **limita Shannon**. [4]

Shannon demonstreaza că pentru orice canal de comunicare, trebuie să existe un cod de corecţie a erorilor care permite transmisiei să se apropie de limita Shannon, dar nu explica cum se poate construi un asfel de cod.

1.1 Coduri corectoare de erori

Un cod corector de erori este un algoritm folosit pentru a exprima o secveță de numere într-o așa manieră încat orice eroare introdusă de zgomotul de pe canal să poată fi detectată și corectată. [5].

Pricipiul pe care se bazează codurile corectoare de erori este în general același: informației îi este adaugătă redundantă, pentru că mai apoi eventualele erori care apar în procesul de transmisie sau stocare să poată fi detectate și corectate. Secvența de biți trimisă pe canal se numește cuvânt de cod, adică simbolurilor de informație li se adaugă simbolurile redundante. Acest tip de codare se numește *codare sistematia* (figura 1.1), adică simbolurile de informație vor apărea întotdeauna în primele k poziţii ale cuvântului de cod. Cele (n-k) simboluri rămase din cuvântul de cod asigură redundanţa, care poate fi folosită pentru detectare/corectare erorilor.

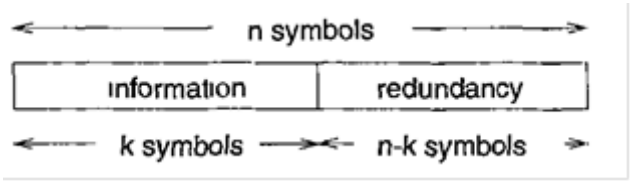


Figura 1.1 Codare bloc sistematică pentru corecţia erorilor

1.1.2 Codurile bloc

Codurile corectoare de erori se împart în două mari categorii din puctul de vedere al cuvintelor de cod: coduri bloc și codurile convoluționale. Codurile bloc proceseaza informația bloc cu bloc, tratand fiecare bloc de biți de informație diferit de celelalte, în timp ce codurile convoluționale fac codarea în mod continuu și există memorie pentru codor. Codarea bloc este o operație fară memorie, adică cuvintele de cod sunt independente unul de celălalt.

În cazul codurilor bloc, toate cuvintele au aceeasi lungime, de exemplu, *n* simblouri 0 și/sau 1 pentru codurile binare. Pentru un cod bloc de lungime *n* cu alfabetul {0, 1}, se pot întocmi 2^n cuvinte de cod, fiecare de lunigime *n.*

Pentru întelegerea codurilor bloc vom defini urmatoarele:

Sursă informaţiei Împărţim o secvenţă cu sursă primară într-un sir de vectori de lungime k înainte de codificare. Un astfel de vecor se notează cu u.

Cod sistematic Un cod în care informația vector apare că o parte a cuvântului de cod sistematic. Segmentul de informație din cuvântul de cod este numit $x_u = u$ și lungimea n-k biţi de paritate adăugaţi x_p.

Proprietatea de liniaritate Codul C este un cod liniar binar dacă şi numai dacă C formează un subspaţiu vectorial peste+. Deci, suma oricăror două cuvinte de cod trebuie să fie în sine un cuvânt de cod.

Dimensiunea codului este dimensiunea spaţiului vectorial care îi corespunde. Un cod binar (n,k) are dimensiunea k, și astfel are un total de 2^K cuvinte de cod, fiecare de lungime n.

Cuvânt de cod diferit de zero O consecinţă directă a proprietăţii de liniaritate este aceea că numele de cod total=zero , x=0, este un membru al oricărui cod liniar.

Coeficientul codului Coeficientul codului este r=k/n. Coeficientul codului indică proporţia de informație transferată pe canal utilizat.

Matricea Generatoare Proprietatea de liniaritate implică existenţa unei baze pentru cod. Să construim o matrice generatoare pentru cod, denumită G, folosind baza independentă de cuvântul de cod k pe post de sir de vectori G. Matricea generatoare are dimensiunea k x n și poate fi folosită pentru a codifica vectorul de informație. Să considerăm forma sistematică $G_{sys} = [P \mid I_k]$ în așa fel încât biții de paritate ai cuvântului de cod să preceadă biții de informație. Aici $[P \mid I_k]$ denotă legatura matricei unitare k x k cu P. Asadar, codificarea este realizată respectând $x = [x_p \mid x_u] = uG$. Matricea generatoare descrie structura codului, totusi nu este definită numai pentru cod.

Matricea Controlului –Parităţii Un alt mod de a descrie structura codului, este furnizat de matricea de control al parității, denumită H. În general, H are dimensiunea m x n, unde m=n-k. Toate cuvintele de cod trebuie să respecte condiţia $Hx = 0$. Matricea de control al parităţii nu este definită numai pentru cod și este legată de G prin $HG^T = 0$. Forma sistematica a lui H poate fi obţinută din G, și viceversă, folosind $H_{sys} = [I_m \mid P^T]$. Matricea H, denumită matrice de control are

numărul de linii egal cu numărul simbolurilor de control și numărul de coloane egal cu lungimea cuvintelor de cod.

Distribuţia greutăţii Definim vectorul de greutate x, W(x), că numărul elementelor diferite de zero pe care le conține. Distribuţia greutăţii unui cod este o listă cu numărul de cuvinte la fiecare greutate, pentru toate greutăţile cuvântului cod.

Distanţa Hamming dintre două cuvinte cod este numărul poziţiilor în care ele diferă.

Distanţa minimă se notează cu d_{min} și este cea mai mică distanţa Hamming dintre două cuvinte de cod din mulţimea tuturor cuvintelor. Având în vedere că un cuvânt de cod nul (cu toti biții zero) este un membru al fiecarui cod liniar, distanţa minimă a unui cod liniar este egală cu greutatea cuvântului de cod care are cea mai mică greutate. În general, un cod cu greutatea minimă d_{min} poate corecta $[(d_{min} - 1)/ 2]$ greseli. [17-lic].

Cuvintele de cod se vor scriu sub forma matriceală:

$[u] = [a_1\ a_2 \ldots a_n]$, $a_i \in \{ 1,0 \}$

Fiind cunoscute cele *k* simboluri de informație, la emisie, codorul determină cele *m* simboluri de control. Pentru simplificarea implementării codorului, simbolurile de control se determină prin combinaţii liniare ale simbolurilor informaţionale, formându-se astfel *codurile bloc liniare*. [6]

II. Codurile LDPC

Codurile LDPC (Low-Density Parity Check) sunt coduri corectoare de erori propuse inițial de catre R. G. Gallager în lucrarea sa de doctorat (1962 MIT). La acea vreme, potențialul lor imens a ramas nedescoperit datorită cerințelor computaționale ale vremii. Ele au ramas în mare parte neglijate timp de 35 de ani. În acest timp în domeniul codurilor corectoare de erori a fost dominat de care codurile convoluționale. [1]

La începutul anilor 1980, Tanner a furnizat o reprezentare grafică a LDPC și a altor scheme de codificare, care sunt cunoscute acum sub denumirea de diagrama Tanner. El a propus ca problema decodificării să fie rezolvată prin descompunerea codurilor în coduri mai simple din structura lor și a introdus noțiunea care este cunoscută în prezent sub denumirea de algoritm de decodificare a sumelor minime.[7]

O data cu descoperirea codurilor turbo de catre Berrou, Galvieux și Thitimajshima în 1993 s-a constatat că se poate tinde catre limita lui Shannon folosind decodoare de o complexitate realizabilă implicând foarte puțina algebră, folosesc algoritmi iterativi, distribuiți, se concentraza pe performața medie (nu performanța cea mai rea).

2.1 Corectarea erorilor folosind verificarea parității

Considerăm un mesaj binar, astfel mesajele transmise constau în siruri de caractere de 0 și 1. Ideea esențială a codurilor corectoare de erori este de a introduce redundanță în mod deliberat sub formă de biți de control suplimentari pentru a produce un cuvânt de cod. Acesti biți de control sunt adaugăti în așa fel incat cuvintele de cod să fie suficient de diferite unele de altele, astefel încât mesajul transmis să se deduca corect la receptor, chiar și atunci cand unii biți din cuvantul de cod au fost schimbati în timpul trasmisiunii pe canal.

Cel mai simpla posibilă schemă de codare este cu un singur bit de paritate. Aceasta presupune adăugarea unui singur bit în plus mesajului binar, a cărui valoare depinde de ceilalți biți din mesaj. Într-un cod de paritate, bitul suplimentar adăugat la fiecare mesaj asigură un număr par de „1” în fiecare cuvânt de cod.

Exemplul 2.1:

Cosideram urmatorul mesaj pe 7 biți u=[1010011], adaugăm bitul de paritate ca al 8-lea bit. Mesajul nostru are deja un numar par de „1”, deci valoarea pentru bitul de paritate va fi *0* și rezultă cuvântul de cod c=[10100110]

La modul general

$$c = [c_1\, c_2\, c_3\, c_4\, c_5\, c_6\, c_7\, c_8],$$

unde c_i este fie 0 sau 1 și fiecare cuvânt de cod trebuie să satisfacă constrangerea:

$$c_1 \oplus c_2 \oplus c_3 \oplus c_4 \oplus c_5 \oplus c_6 \oplus c_7 \oplus c_8 = 0. \qquad (1.1)$$

Ecuatia (1.1) se numește ecuație de verificare a parității unde simbolul $\oplus$ reprezintă suma modulo 2.

Cuvantul de cod din exemplul de mai sus este trimis pe un canal cu zgomot și se receptioneaza y = [10010010]. Pentru a verifica dacă y este un cuvânt de cod valid îl testăm cu (1.1)

$$y1 \oplus y2 \oplus y3 \oplus y4 \oplus y5 \oplus y6 \oplus y7 \oplus y8 = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 1.$$

Se observa că suma este 1, deci nu este satisfacută ecuația de paritate prin urmare y nu este un cuvânt de cod valid. Am detectat că avem cel putin o eroare în timpul transmisiei.[8]

Detecția inversiunii unui singur bit dintr-un cuvânt de cod se face cu usurință folosind un singur bit de paritate, totuși acest cod nu este suficient de puternic pentru a indica care bit/biți au fost inversați. Mai mult, dacă un numar par de biți sunt inversați, atunci se satisface ecuația (1.1), iar perechile de erori ce pot surveni nu pot fi detectate de acest cod simplu. Pentru a detecta mai mult de o eroare avem nevoie de mai multă redundanță și un cod mai sofisticat cu mai multe ecuații de paritate.

Exemplul 2.2:

Consideram un cuvânt de cod de forma: $c = [c_1\ c_2\ c_3\ c_4\ c_5\ c_6]$, care satisface toate trei ecuații de paritate descrise mai jos:

$$\begin{aligned} c1 \oplus c2 \oplus c4 &= 0 \\ c2 \oplus c3 \oplus c5 &= 0 \\ c1 \oplus c2 \oplus c3 \oplus c6 &= 0 \end{aligned} \qquad (1.2)$$

Ecuațiile de paritate se pot scrie și sub forma matriceală:

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}}_{H} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \qquad (1.3)$$

Matricea *H* se numește matrice de verificare a parității. Fiecare rand din H corespunde cu o ecuație de verificare a parității, iar fiecare coloană din H corespunde cu un bit din cuvântul de cod. În forma matriceală, un șir recepționat $y = [c_1\ c_2\ c_3\ c_4\ c_5\ c_6]$ este un cuvânt de cod valid dacă satisface ecuația:

$$Hy^T = 0. \qquad (1.4)$$

2.2 Coduri Low Density Parity Check

După cum sugerează și numele, codurile LDPC sunt coduri bloc ale căror matrici de verificare a parității conțin un numar foarte mic de „1”. Aceasta matrice „rară” asigură atât complexitatea la decodare cât și distanța minimă vor creste liniar.

Spre deosebire de celelalte coduri bloc, codurile LDPC sunt construite astfel încât inițial se construiește matrice *H* apoi se construiește matricea generatoare.

Cea mai mare diferenta dintre codurile LDPC și codurile bloc clasice o reprezintă modalitatea de decodare. Codurile bloc clasice sunt în general decodate cu algoritmi de tipul ML (maximum-likelihood - probabilitate maximă), deci sunt de obicei scurte și proiectate pentru o complexite algebrică scazută. Codurile LDPC sunt decodate iterativ folosind reprezentarea grafica a matricii de paritate și sunt proiectate având ca punct de plecare proprietățile matricii de paritate. Numarul de elemente de „1” de pe coloană se numește greutatea coloanei și se noteaza în general cu w_c, respectiv, numarul de „1” de pe rand se numește greutatea randului și se noteaza cu w_r.

Gallager definește o structura regulata pentru codurile LDPC după cum urmeaza: ***un cod regulat LDPC cu (n, j, i) este un cod bloc de lungime n care are o matrice de paritate cu greutăți pe rand și pe coloană de exact j pe coloană și exact i pe coloană.*** [1]

Prin contrast, codurile neregulate LDPC permit variabilelor să participe în numere diferite de controale și fac posibilă aplicarea restricţiilor controlului la numere diferite de variabile. Un cod neregulat LDPC poate fi descris de distribuţiile greutăţii vectorului de linie și de coloană ale matricii sale de control.

2.3 Codarea LDPC

Codarea informației în cazul codurilor LDPC depinde în mare masură de costruirea matricii de verificare a parității că fiind una cu puține valori de „1" intr-o matrice de zerouri astfel să fie respectate anumite conditii.

Versiunea prezentată inițial de Gallager în lucrarea sa a fost o versiune de coduri LDPC regulate cu o matrice de control definită după o structura cu benzi. Rândurile matricii lui Gallager sunt împărțite în w_c seturi cu M / w_c rânduri în fiecare set (M = numărul de rânduri). Primul set de rânduri conțin un numar de „1" consecutivi egal cu w_r ordonați de la stanga la dreapta pe fiecare coloană. Fiecare din restul de seturi de rânduri rămase sunt alese în mod aleatoriu prin permutarea coloanelor primului set. Fiecare coloană are în mod consecutiv o singură valoare de „1" în fiecare din cele w_c seturi.

Exemplul 2.3:

O matrice de control al parității după modelul lui Gallager de lunigime 12:

$$H = \left[\begin{array}{cccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right].$$

În general codurile LDPC sunt construite în mod aleatoriu, deci se urmarește în această construcție respectarea unui set de reguli.

Pentru o codarea LDPC se folosește o matrice de forma $H = [A, I_{n-k}]$, unde A este o matrice binara de dimenisuni $(n - k)$ x k, iar I_{n-k} este matricea unitate.

Prin urmare matricea generatoare v-a fi de forma $G = [I_k , A^T]$.

Pentru a distinge mai bine intre biții de informație și biții de paritate din cuvantul de cod, rescriem ecuațiile de paritate din exemplul 2.2 astfel:

$$\begin{aligned} c_4 &= c_1 \oplus c_2 \\ c_5 &= c_2 \oplus c_3 \\ c_6 &= c_1 \oplus c_2 \oplus c_3 \end{aligned} \qquad (1.10)$$

Biti c_1, c_2 și c_3 conțin biții de informație iar biții c_4, c_5 și c_6 conțin biții de paritate. Scrise în această formă, ecuațiile de paritate ne arată cum se codeaza un mesaj.

Exemplu 2.4

Folosind ecuațiile (1.5) mesajul 110 produce biții de paritate:

$$c_4 = 1 \oplus 1 = 0,$$

$$c_5 = 1 \oplus 0 = 1,$$

$$c_6 = 1 \oplus 1 \oplus 0 = 0,$$

rezultă cuvantul de cod pentru mesajul 110 c = [1 1 0 0 1 0].

Ecuatiile de mai sus pot fi scrise și sub formă matriceală după cum urmează:

$$\begin{bmatrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}}_{\mathbf{G}}, \qquad (1.6)$$

unde G este numita matricea generatoare a codului. Biții de informație sunt etichetați cu **u** = [u_1, u_2, ...u_k], unde vectorul **u** conține cei k biți de informație. Cuvantul de cod **c** corespunde mesajului binar u = [$u_1u_2u_3$] se formează folosind ecuația matriceală

$$\mathbf{c} = \mathbf{u}G. \qquad (1.7)$$

Un cod care are *k* biți de informație conține 2^k cuvinte de cod. Inlocuind cele $2^3 = 8$ mesaje dinstincte $c_1c_2c_3$ = 000, 001, ..., 111 în ecuația (1.7) obținem urmatorul set de cuvinte de cod:

[0 0 0 0 0 0] [0 0 1 0 1 1] [0 1 0 1 1 1] [0 1 1 1 0 0]

[1 0 0 1 0 1] [1 0 1 1 1 0] [1 1 0 0 1 0] [1 1 1 0 0 1] (1.8)

Codul se numește sistematic pentru că primii k biți ai cuvantului de cod reprezintă biții de informatiție. În codurile sistematice, matricea generatoare conține matricea unitate de dimensiuni *k* x *k*, I_k. [18]

2.3.1 Detecția și corectarea erorilor

Presupunem că mesajul codat mai sus a fost trimis pe un canal cu zgomot și unul sau mai mulți biți din cuvantul de cod au fost schimbați. Urmărim în continuare să detectăm biții eronați și dacă e posibil să îi corectăm. Știm că fiecare cuvânt de cod trebuie să sătisfacă ecuațiile de paritate (1.4), astfel încât erorile vor fi detectate dacă ecuațiile nu sunt satisfăcute.

Exemplul 2.5

Cuvântul de cod c = [1 0 1 1 1 0] din exemplul anterior se trimite pe un canal afectat de zgomot, iar la recepție avem sirul y = [1 0 1 0 1 1]. Calculăm sindromul după formula (1.4).

$$H\mathbf{y}^T = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \qquad (1.9)$$

Observăm că rezultatul este diferit de *0*, deci concluzionăm că nu avem un cuvânt de cod valid adică ecuațiile de verificare a parității nu sunt satisfăcute. În acest caz, valoarea sindromului ne indică faptul prima ecuație da paritate nu este sătisfăcută de *y*. Prima ecuație de paritate implică biții 1, 2 și 4 concluzionăm că unul dintre acesti trei biți au fost inversați.

În exemplul de mai sus observam că sirul recepționat *y* = [1 0 1 0 1 0] nu este un cuvânt de cod din exemplul 2.4. Prin compararea lui *y* cu fiecare cuvânt de cod (1.8), un decodor de tipul Probabilitatii Maxime (**ML**) ar alege **c** = [1 01 1 1 1 0] că fiind cel mai apropiat cuvânt de cod și de altfel singurul cuvânt de cod care are distanța Hamming 1 față de **y** [19][20].

2.3.1 Reprezentarea Tanner a codurilor LDPC

Codurile LDPC sunt deseori reprezentate prin graful lui Tanner [7]. Acest graf constă în două seturi de noduri: *n* noduri pentru biții cuvantului de cod (numite și nodurile biților sau bit-node), și *m* noduri pentru ecuațiile de verificare a parității (numite și noduri de paritate sau check-node). O muchie a grafului uneste un bit-node cun un check-node dacă acel bit este inclus în ecuația de paritate corespunzatoare rezultă deci că numarul de muchii din graful lui Tanner este egal cu numarul de „1" din matricea H.

Exemplu 2.6:

Fie matricea de mai jos:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Graful Tanner pentru matricea de mai sus:

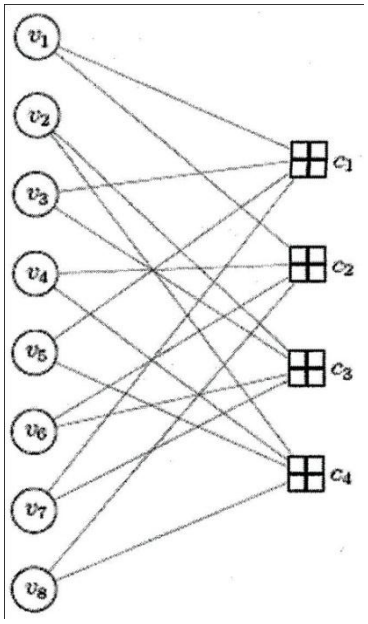


Figura 2.1 Graful Tanner pentru matricea de paritate

Se pot deduce ecuațiile de verificare a parității după cum urmeaza:

$$v_1+v_3+v_5+v_7=c_1$$

$$v_1+v_4+v_6+v_8=c_2$$

$$v_2+v_3+v_6+v_7=c_3$$

$$v_2+v+v_5+v_8=c_4$$

2.3.3 Transformarea matricii H

După structurile de matrici propuse de Gallager, Makay s.a, Sipser și Spielman au propus o clasă de coduri de expansiune decodabilă și codificate în timp liniar. Codurile corectoare de erori sunt construite combinând recursiv coduri mai slabe de reducere a erorii. Au dovedit că în cazul în care codurile de reducere a erorii sunt codificabile/decodificabile în timp liniar, această proprietate va fi păstrată până la obţinerea codului final corector de erori [10]. Luby s.a. au construit coduri bazate pe aceste structuri care au o performanţă foarte bună [11].

Echard și Chang au prezentat o variantă de construire a codurilor LDPC srtucturate cvasi-regulat, numită codurile de rotaţie [12,13]. Matricea de control al parităţii pentru aceste coduri este construită din componente ale matricelor de permutare și rotaţiile lor.

O altă variantă de a asigura codificarea liniară de timp foloseste structurile ciclice și cvasi-ciclice. Un cod ciclic are proprietatea că orice deplasare ciclică a unui cuvânt de cod este în sine un cuvânt de cod. Un cod cvasi-ciclic are proprietatea că, pentru mărimea fixă de deplasare, o deplasare ciclică a oricărui cuvânt de cod de acea mărime este în sine un cuvânt de cod. Putem construi matricea de control al parităţii pentru un cod LDPC cvasi-ciclic prin unirea pe orizontală a matricelor rare circulare, fiecare având dimensiunea *m x m.*

Definiţia 2.6 (Matricea Circulară) O matrice circulară este o matrice pătrată în care fiecare sir este o deplasare ciclică a sirului anterior.

Un exemplu de matrice de control al parităţii pentru un cod cvasi-ciclic având n=18 și k=12, și deci r=2/3 , este dat în figura 2.8. Această matrice poate fi rearanjată cu usurinţă în forma cvasi-ciclică prezentată prin permutarea coloanelor săle în ordinea 1, m + 1, 2m + 1, 2, m + 2, 2m + 2, ...m, 2m, 3m. Se observă că permutarea liniilor și coloanelor H nu schimbă structura codului ci doar redenumește controalele sau respectiv variabilele.

$$H = \left[\begin{array}{cccccc|cccccc|cccccc} 1&1&0&1&0&0&1&0&1&0&1&0&0&0&1&0&1&1\\ 0&1&1&0&1&0&0&1&0&1&0&1&1&0&0&1&0&1\\ 0&0&1&1&0&1&1&0&1&0&1&0&1&1&0&0&1&0\\ 1&0&0&1&1&0&0&1&0&1&0&1&0&1&1&0&0&1\\ 0&1&0&0&1&1&1&0&1&0&1&0&1&0&1&1&0&0\\ 1&0&1&0&0&1&0&1&0&1&0&1&0&1&0&1&1&0 \end{array}\right]$$

H_p | H_{u1} | H_{u2}

Figura 2.2: O matrice cvasi-ciclică de control al parităţii

Împărţim H în trei componente circulare *m x m*. Aici există un total de componente ale matricei n_0=3, iar k_0=2 din acestea corespunde biţilor de informare ai cuvântului de cod. Mai exact, H_p corespunde biților de paritate, în timp ce H_{u1} și H_{u2} corespunde biților de informare. Dacă H_p nu este singular atunci putem începe prin a multiplica H cu H_p^{-1} pentru a obţine forma sistematică $H_{sys} = [I_m|_{Hu(sys)}] = [I_m \mid H_p^{-1}H_{u1} \mid H_p^{-1}H_{u2}]$. Apoi permutăm coloanele aparţinând $H_{u(sys)}$ în ordinea 1, m + 1, , 2, m + 2 ...m, 2m pentru a redefini biții de informare corespunzător. Acest loc apare într-o formă sistematică cvasi-ciclică după cum e arătat în figura 2.9.

Se observă că fiecare sir aparţinând $H_{u(sys)}$ este o deplasare ciclică corectă, prin locurile k_0. Acest lucru permite codurilor cvasi-ciclice să fie codificate folosind aceeasi metodă că aceea folosită de codurile ciclice. Pentru a genera biții de paritate pentru codul de mai sus folosim

arhitectura registrului de deplasare regresivă a stadiului k din figura 2.10. În această diagrama vectorul sursă, u, este luat din stânga și plasăt în dreapta,

$$\mathbf{H}_{sys} = \left[\begin{array}{cccccc|cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \end{array}\right]$$

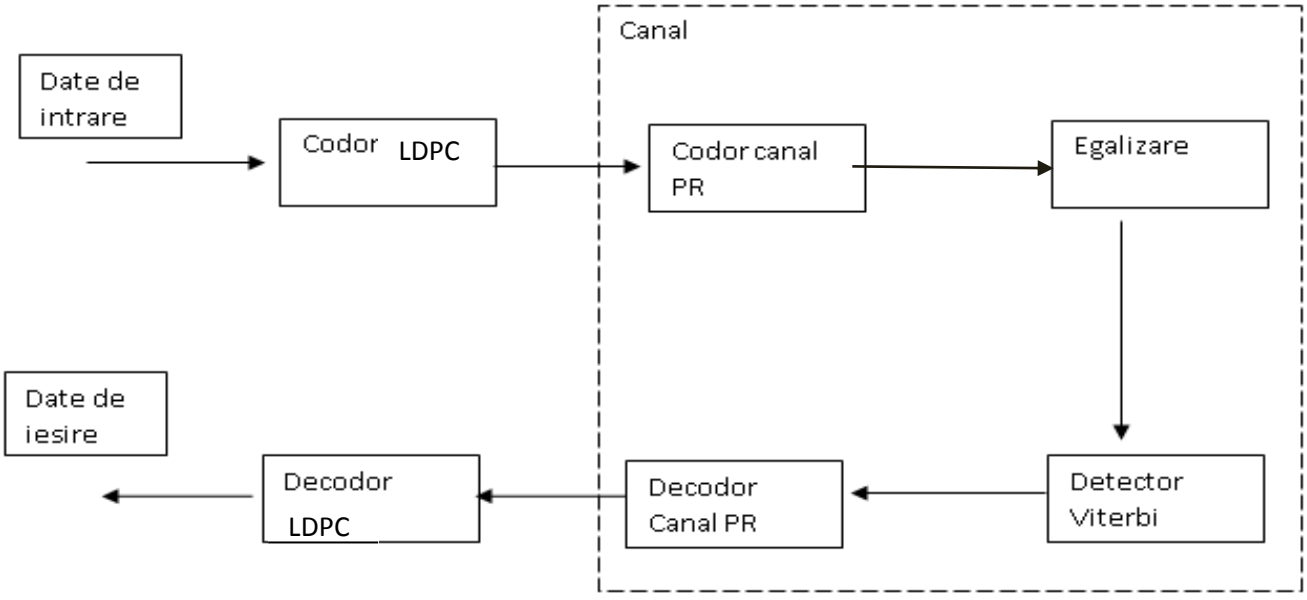
$\mathbf{I}_m$ $\mathbf{H}_{u(sys)}$

$x_{p1} \quad \cdots \quad x_{p6} \quad x_{u1} \quad \cdots \quad x_{u12}$

Figura 2.3: Forma sistematică cvasi-ciclica a H

începând cu bit-ul u_I. Fiecare cutie reprezintă o celulaă de memorie care păstrează valoarea încărcată de obicei până când este aplicată o deplasare, stadiu în care ia valoarea celulei precedente. Schimbătorul s este inițial conectat la o sursă de informație în serie, pentru a încărca vectorul sursă în registru. Această operaţie are loc în k deplasări. Primul bit de paritate este apoi calculat printr-o operaţie XOR pe biții de informație selectati, alesi de ramurile paralel ale codorului cu registru de deplasare. Poziţiile ramurilor sunt setate pe poziția (inversată) de termeni zero în primul sir de $H_{u(sys)}$. Schimbătorul este apoi închis în poziţia inițiată și datele sunt deplasăte ciclic de k_0 ori. Apoi următorul bit de paritate este calculat. Acest proces este repetat până când toti biții de paritate m vor fi generaţi.

2.4 Prezentarea unui model al sistemelor de comunicaţie



Un sistem de comunicaţie complet include numeroase părţi componente care pot ridica probleme interesănte și greu de rezolvat. Cele mai multe sisteme de comunicaţie moderne lucrează în domeniul digital, acesta oferind o mulţime de posibilităţi de procesăre a semnalului. În general, pentru obţinerea de performanţe optime, codorul sursei, codorul de canal și modulatorul ar trebui considerate că făcând parte din aceeasi funcţie a sistemului și nu că blocuri separate. Shannon a demonstrat că partea de codare a sursei și partea de codare de canal pot fi separate în unităţi individuale fără a se pierde la capitolul optimizare.

Aceeasi regulă însă nu se poate aplica și pentru despărţirea funcţiilor de codare de canal și modulare. De exemplu, modulaţia bazată pe codare trellis(TCM), care este o tehnică ce combină

codarea de canal şi modulaţia, poate obţine performanţe mai ridicate decât metodele care separă codarea de canal şi modulaţia.

2.4.1 Modele de canal

Pentru studierea şi compararea codurilor de canal este folositor să considerăm modulatorul şi demodulatorul că fiind părţi componente ale canalului. Rezultatul este un canal cu intrări şi iesiri în timp discret, caracterizat de intrările şi iesirile posibile şi de probabilităţile de tranziţie care fac legătura dintre intrări şi iesiri. De asemenea, iesirea poate să nu depindă numai de intrarea curentă ci şi de cele anterioare. În aceste cazuri vorbim despre canale cu memorie.

2.4.1.1 Canalul AWGN

Acest proiect are că scop înţelegerea şi simularea codurilor LDPC. Îacest scop va fi folosit canalul cu zgomot aditiv Gaussian alb(additive white Gaussian noise - AWGN), acesta fiind un tip de canal cu o bună relevanţă practică. Asa cum este sugerat şi de numele canalului iesirea Y este modelată prin adăugarea unei variabile aleatoare distribuite Gaussian (G) la intrarea în canal X.

$$Y=X+G \qquad (1.11)$$

G este o variabilă aleatoare distribuită Gaussian de medie zero şi varianţă a_2.

Pentru un anumit simbol X= x iesirea din canal Y este o variabilă aleatoare distribuită Gaussian de medie x şi varianţă σ^2, dată de formula:

$$P_{Y|X}(y \mid x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y-x)^2}{2\sigma^2}} \qquad (1.12)$$

unde p( ) reprezintă densitatea de probabilitate a unei funcţii de o variabilă aleatoare.

2.4.2 Limite de performanţă

Ca parte a teoriei matematice a comunicaţiei, Shannon a definit conceptul de capacitate de canal. Capacitatea unui canal reprezintă o măsura a cantităţii de informaţie ce poate fi transportată între intrarea X şi iesirea Y a unui canal. Definiţia capacităţii este legată de definiţia matematică a informaţiei; informaţia medie mutuală între variabilele aleatoare continue X şi Y , în biţi, definite astfel:

$$I(X,Y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} px,y(x,y)\log_2 \frac{p_{x,y}(x,y)}{p_x(x)p_y(y)}dxdy \qquad (2.3)$$

unde Px,y(x,y), $p_x(x)$ şi $p_Y(y)$ sunt densitatile de probabilitate pt X şi Y.

Capacitatea canalului C este definită că maximul valorii I(X,Y) pentru distribuţia de intrare $p_x(x)$ măsurată în biţi/canal

$$C = \max I(X;Y) \qquad (2.4)$$

Teorema codării(Shanon) 2.1 de canal asigură o relaţie între capacitatea canalului C şi rata de transmisie a informației R:

Dacă avem o sursă cu un debit de informaţie R biţi/s şi un canal de capacitate C biţi/s şi dacă R < C, există un cod cu cuvinte de lungime n astfel încât probabilitatea unei erori de decodare P_E să fie:

$$P_E \le 2^{-nE(R)} \qquad (1.13)$$

unde n este lungimea cuvântului de cod şi E(R) o funcţie nenegativă numită exponentul erorii.

Deci, indiferent de nivelul perturbaţiilor din canal se poate face transmisiunea cu o probabilitate a erorii oricât de mică.

Performanţele codorului LDPC sunt foarte bune tinzând spre limita teoremei lui Shannon. Se defineste eficienţa spectrală: $ES = r_b / B$ bit/sec/hz, unde r_b este rata de transmisie şi B banda .

Eficienţa spectrală maximă

$$ES_{\max} = \log_2 MR \tag{1.14}$$

Raportul semnal-zgomot

$$\frac{S}{N} = \log_2 MR \frac{E_b}{N_0} \tag{1.15}$$

unde M este modulaţia, R rata codorului şi $\frac{E_b}{N_0}$ reprezintă raportul dintre energia de bit și densitatea spectrală de putere a zgomotului.

Teorema lui Shannon spune că se poate realiza o probabilitate de eroare de bit oricât de mică prin codare-decodare dacă rb < C, unde C este capacitatea canalului

$$C = B \log_2 (1 + \frac{S}{N}) \tag{1.16}$$

Dacă $r_b = C$ (1.17)

Rezultă că eficienţa spectrală maximă este:

$$ES_{\max} = \log_2 (1 + ES_{\max} \frac{E_b}{N_0}) \Rightarrow \frac{E_b}{N_0} = \frac{2^{ES_{\max}} - 1}{ES_{\max}} \tag{1.18}$$

reprezintă raportul minim dintre energia de bit şi densitatea spectrală de putere a zgomotului necesăr pentru transmisie.

Pentru canalul discret AWGN cu intrări şi iesiri continue capacitatea canalului este dată de relaţia :

$$C = \frac{1}{2} \log_2 (1 + \frac{\sigma^2{}_X}{\sigma^2}) \text{ biţi/canal} \tag{1.19}$$

unde puterea la intrarea canalului este limitată de $E[X^2] \le \sigma^2{}_x$ şi σ^2 este varianţa zgomotului.

În practică forma de undă folosită este limitată de bandă. Deci rata de transmisie nu poate fi crescută oricât prin accesărea oricât de des a canalului.

O cerinţă fundamentală pentru comunicaţie este dată de:

$$\frac{E_b}{N_0} > \frac{1}{\log_2 e} = \ln 2 \tag{1.20}$$

Deci nu este posibil să se conceapă un sistem cu o rată arbitrară scăzută de erori dacă Eb/N_0 < 0.7 dB . Pe de altă parte, atât timp cât condiţia este îndeplinită, teoretic se pot atinge probabilităţi de eroare scăzute folosind coduri suficient de mari.

Această restricţie impune o limită inferioară în ceea ce priveste raportul energie per bit pe densitatea spectrală a zgomotului (Eb/N_0), limită ce trebuie respectată pentru a putea obţine o comunicaţie fără erori. Oricum, presupunerile folosite pentru a obţine aceasta limită sunt prea optimiste pentru cele mai multe situaţii reale. Cea mai importantă diferenţă între descriererea teoretică şi realizarea practică este dată de faptul că banda canalului este, în realitate, limitată. Din această cauză raportul Eb/N_0 necesăr pentru o comunicaţie bună, chiar folosind coduri foarte mari, este mai mare decât 0.7 dB.

2.4.1.2 Structura fizică a unui hard-disc

În această secţiune vom analiza pe scurt componentele de bază ale unui hard-disc şi vom evidenţia influenţele acestora asupra semnalului înregistrat acolo unde este necesăr.

Figurile următoare prezintă o imagine simplificată a componentelor mecanice care alcătuiesc sistemul de înregistrare magnetică al unui hard-disc.

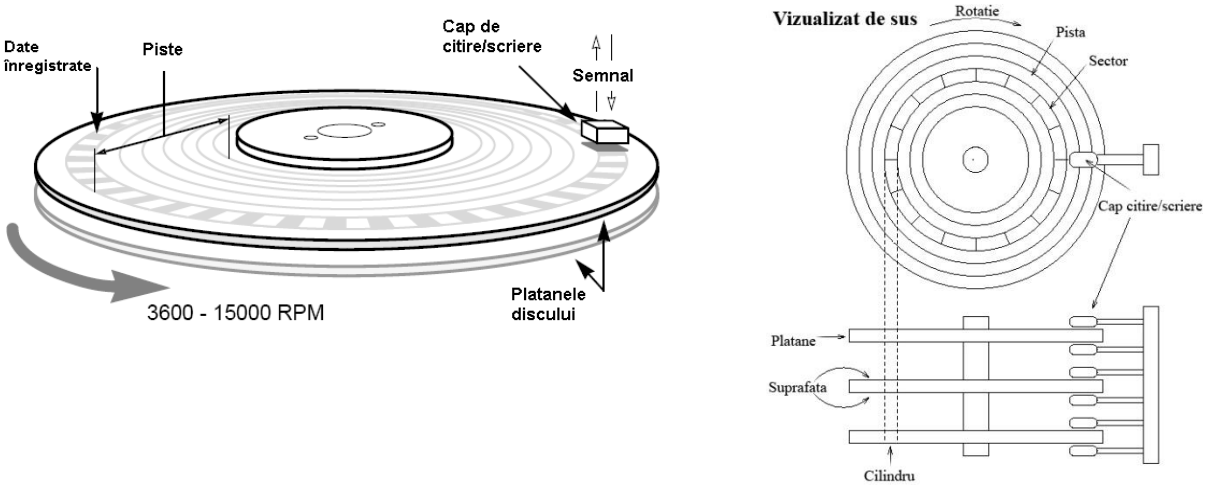


Figura 2.4: Componentele mecanice esenţiale ale unui hard-disc.

Există câte un cap de citire/scriere suspendat deasupra fiecărei suprafeţe (platan) al unui disc rotativ, pe care sunt stocate datele. Pe măsură ce discul se roteste, capul va trece peste o regiune inelară a discului numită **pistă** unde va fi făcută înregistrarea propriu-zisă. Dacă discul este constituit din mai multe platane, atunci pistele cu aceeasi rază de pe fiecare platan vor forma un **cilindru**. Discul este învelit într-o peliculă de material magnetic dur, care va reţine magnetizarea creată de capul de citire/scriere. Datele sunt înregistrate sub forma unor regiuni magnetizate sau celule bit, pe fiecare pistă. Fiecare bit înregistrat va avea una din cele două direcţii posibile de magnetizare ale câmpului magnetic. Trecerea de la o direcţie de magnetizare la cealaltă poartă denumirea de **tranziţie magnetică.**

O formă de undă tipică a tensiunii induse în capul magnetic de citire, este prezentată în figura 1.4. Trebuie menţionat că în figură s-a considerat semnalul că fiind neafectat de zgomot şi că de obicei înainte de a fi scrise, datele sunt precodate, astfel că în timpul procesului de citire un puls

de tensiune pozitiv sau negativ va corespunde unui bit de „1”, în timp ce absenţa unui puls va corespunde unui bit de „0” (codare **NRZI**).

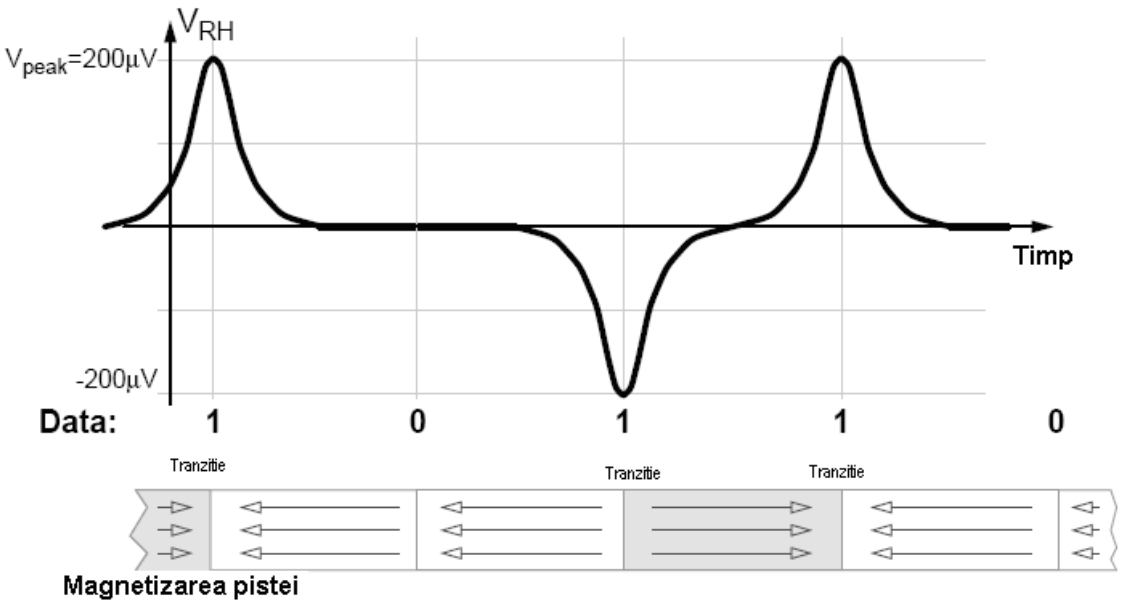


Figura 2.5: Forma tensiunii induse în capul de citire (la densitate de înregistrare scăzută).

Din figura anterioară sunt importante de reţinut două aspecte: unui bit de „1” îi va corespunde întotdeauna o tranziţie magnetică și două pulsuri de tensiune consecutive vor avea întotdeauna polarităţi diferite. Perioada de timp în care amplitudinea unui puls de tensiune depăseste 50% din valoarea să de vârf este considerată că reprezentând 50% din lăţimea impulsului și se notează cu **PW50.** Această noţiune este importantă deoarece densitatea de înregistrare a canalului magnetic se defineste că fiind raportul dintre PW50 și perioada de bit a canalului (**T**): **D=PW50/T.** La densităţi de înregistrare foarte mari spaţiul dintre impulsurile de tensiune din figura 1.4 începe să scadă până când acestea vor începe să se suprapună, fenomen care poartă denumirea de **interferenţă intersimbol (ISI).** Prezenţa ISI împreună cu cea a zgomotului poate conduce la nivele foarte ridicate ale ratei de erori în timpul procesului de citire/recuperare a datelor.

2.4.1.2 Metode de codare a informației în canale magnetice

Codarea informației are un rol esenţial în structura canalului magnetic de înregistrare, în acest mod prevenindu-se probleme majore care pot afecta sincronizarea capului de citire-scriere cu discul magnetic și integritatea datelor.

În figura următoare sursă de informaţii este de obicei un sistem de calcul sau un alt sistem capabil să genereze date în format binar. Informaţia este codată cu ajutorul unui cod corector de erori (de exemplu cod Reed-Solomon) cu rol de protecţie a informației stocate în cazul unor defecte de material, zgomote, etc. Semnalul binar rezultat este în continuare aplicat unui circuit de scriere care realizează prelucrarea acestuia în vederea adaptării săle la mediul magnetic de stocare. Aceste prelucrări constau în codări suplimentare (de exemplu codare RLL, etc), amplificare, etc. Semnalul obţinut este apoi aplicat capului de scriere, care realizează înregistrarea fizică a informației pe suportul magnetic, sub forma unor tranziţii în magnetizarea acestuia.

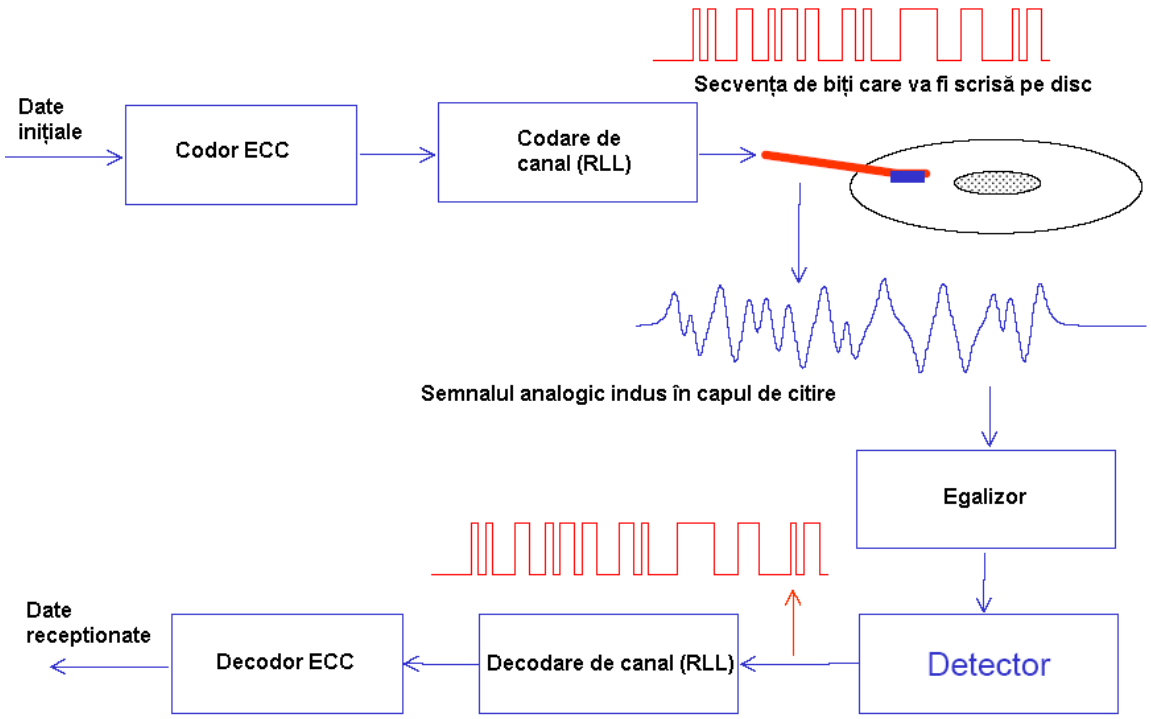


Figura 2.6: Schema tipică a unui canal de înregistrare pe suport magnetic.

La citire, capul magnetic de citire furnizează în exterior un semnal obţinut prin inducerea în circuitul magnetic al acestuia a unei tensiuni prin inducţie electromagnetică. Acest semnal este aplicat unui circuit care realizează o amplificare și eventual alte prelucrări (de exemplu, filtrarea în vederea eliminării interferenţei între biții vecini). Semnalul (analogic) obţinut la iesirea circuitului amintit mai sus este aplicat unui circuit de detecţie, care realizează refacerea informației digitale. Informaţia digitală obţinută este decodată, corectându-se eventualele erori apărute, și furnizată sistemului de calcul.

În continuare vom face o serie de precizări suplimentare legate de codurile RLL. Am precizat anterior faptul că înainte de a fi scrisi, biții de informație sunt precodaţi, sau codaţi NRZI. Codul NRZI este folosit pentru că dispozitivul de citire (capul magnetic) are o caracteristică de derivator (funcţionează pe baza legii inducţiei electromagnetice), deci în semnalul de iesire tranziţiile de magnetizare vor corespunde unor vârfuri, care pot fi detectate cu usurinţă că biți de “1”. Principalul dezavantaj al acestui cod este faptul că la apariţia unor siruri lungi de biți “0” este posibilă pierderea sincronizării circuitului de citire. Mai precis, datorită unor fluctuaţii inerente în viteza de rotaţie a discului, capul magnetic ajunge să fie poziţionat incorect în raport cu locaţia de pe disc, care trebuie citită/scrisă. Aceste fluctuaţii sunt compensăte de un circuit specializat, o buclă PLL, care însă are la rândul ei nevoie de un semnal de tact pe care îl preia chiar din semnalul indus în capul de citire de secvenţa înregistrată. Un sir lung de biţi de „0” va duce la absenţa acestui semnal, așa cum se poate observa în figura 1.6, și la imposibilitatea compensării fluctuaţiilor de viteză amintite mai sus. Codurile RLL previn această problemă introducând câte un bit de „1” după un anumit număr de biţi „0”.

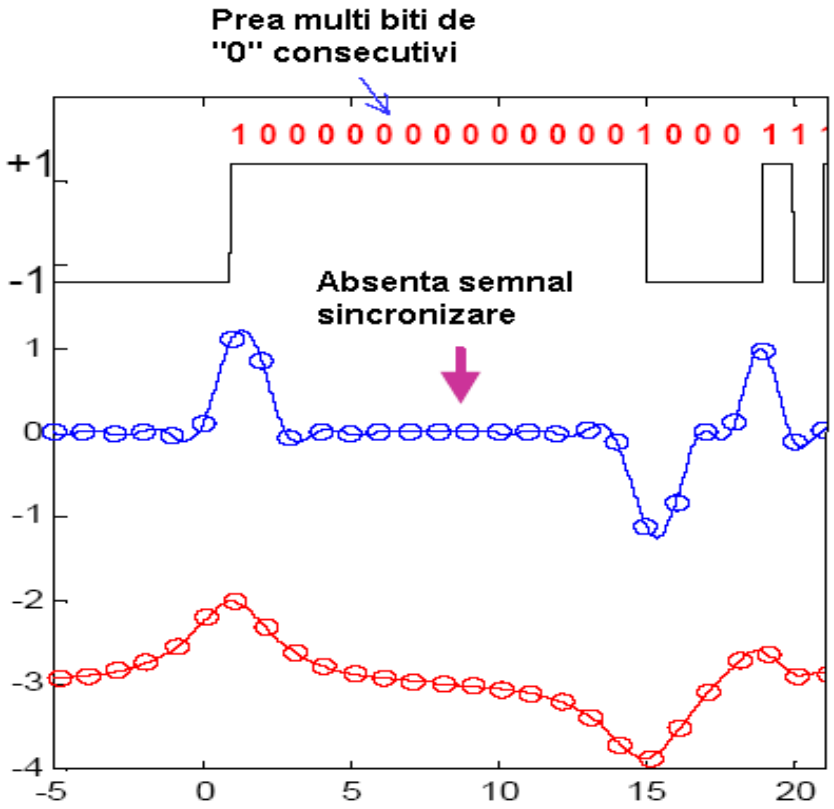


Figura 2.7: Forma de undă asociată unei secvenţe lungi de biți „0”.

Codurile RLL clasice, (de exemplu **RLL** (1,7) și **RLL** (2,7)) au că principal dezavantaj introducerea unei redundanţe semnificative a secvenţei codate în raport cu secvenţa originală (între 50%-100%). Acest fenomen implică necesitatea cresterii densităţii de înregistrare pentru a compensă spaţiul pierdut prin codare şi astfel se pierde câstigul iniţial obţinut prin micsorarea ratei de erori de bit. Codurile **MTR (Maximum Transition Run)** RLL, rezolvă această problemă prin adaptarea tehnicii de codare la tipul canalului. Astfel, deoarece canalele PRML și DFE permit nativ densităţi de înregistrare mai ridicate, sau altfel spus sunt mai puţin sensibile (DFE) sau chiar utilizează efectele ISI (PR), codul nu mai este obligat să separe perechile de biţi de ‚1’ prin biţi de ’0’, ci poate permite secvenţe de tranziţii magnetice de lungime maximă fixată [12]. Pentru codurile din această lucrare, lungimea maximă este de 3. În acest mod redundanţele efective ale codurilor implementate variază între: 25% și 6,25%, reducând spaţiul pierdut pe disc prin codare.

2.5 Decodarea codurilor LDPC

Algoritmii folositi pentru decodarea codurilor LDPC sunt numiți colectiv algoritmi cu trasmitere de mesaje (messăge-passing), nume ce poate fi explicat prin operatiile de trimitere a mesăjelor pe muchiile grafului lui Tanner. Acesti algoritmi sunt denumiți după tipul de mesaje transmise sau după tipul de operatii care se execută la fiecare nod.

2.5.1 Decodarea iterativă

Algoritmii de decodare cu transmitere de mesaje sunt cunoscuti deasemenea că și algoritmi de decodare iterativi pentru că mesajele se transmit înainte și înapoi pe muchiile grafului lui Tanner intre nodurile de verificare a parității și nodurile biților de informație iterativ pană când rezultul este atins sau procesul este oprit.

2.5.2 Algoritmi de decodare LDPC

În unii algoritmi, cum ar fi bit-fliping, mesajele transmise sunt binare, în altele cum ar fi belief propagation (propagarea increderii), mesajele sunt probabilitati care reprezintă nivelul de incredere asuprea valorilor biților cuvntului de cod. Pentru că deseori este mai convenabila reprezentarea probabilitatilor cu ajutorul logaritmilor, algoritmul belif propagation se mai numește și algoritmul sumă-produs, iar pentru că raportul probabilitatilor se face folosind operații de sumă și produs.

2.5.3 Decodarea Hard-Decision – algoritmul bit – flipping

Algoritmul bit-fliping este un algoritm hard-decision de trasmitere a mesăjelor pentru codurile LDPC. Pentru algoritmii de tip hard-decision, mesajele trimise pe muchiile grafului lui Tanner sunt valori binare: un bit-node trimite un mesaj prin care declara valoarea să, *0* sau *1*, iar fiecare check-node trimite mesaje catre fiecare bit-node conectat, cu valoarea determinata în funcție de informația aflată check-node. Check-node determină dacă ecuația de paritate este sătisfacuta, dacă suma modulo 2 a biților primiți este 0. Dacă majoritatea mesăjelor primite de catre bit-node sunt diferite de valorea primită inițial valoarea bit-node se schimba (flips) în valoarea curent majoritară. Acest proces se repeta pană cand ecuațiile verificare a paritatății sunt satisfăcute sau pana cand un numar maxim de iteratii a fost atins.

Decodarea se poate termina imediat cand un cuvânt de cod valid a fost gasit prin verificarea tuturor ecuațiilor de verificare a parității, de aici putem deduce două avantaje importante ale decodarii prin transmiterea mesăjelor a codurilor LDPC: primul este că putem evita iterații suplimentare în momentul în care s-a gasit o solutie și în al doilea rand incapacitatea de a ajunge la un cuvand de cod valid este tot timpul detectată.

Algoritmul bit-flipping este bazat pe principiul că un bit al unui cuvânt de cod implicat intr-un numar mare de ecuații de verificare a parității incorecte este probabil că el insusi să fie incorect. Faptul că matricea ***H*** este slab populată imprastie biții în ecuațiile de verificare a parității astfel incat este putin probabil că să contina același set de biți ai cuvantului de cod. [8]

Exemplul 2.7:

În [17], Leiner foloseste un cod bloc liniar (4, 8) pentru a ilustra decodeor hard-decision. Codul este reprezentat de figura ‚2', având urmatoare matrice de paritate:

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

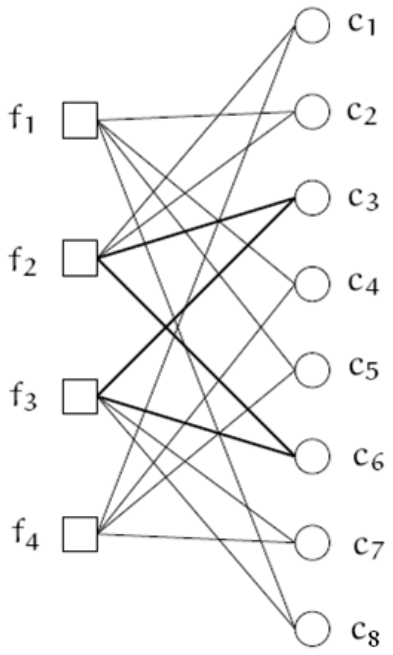


Figura 2.8 Graful Tanner pentru matricea de mai H

Un cuvânt de cod c = [1 0 0 1 0 1 0 1] este transmis pe canal. Presupunem că recepționăm y = [1 1 0 1 0 1 0 1], prin urmare c_2 a fost schimbat.

1. În primul pas toate nodurile bit-nodes (c_i) trimit un“mesaj” catre nodurile lor check-nodes f_j (intodeauna două în cazul nostru) continand bit-ul care il considera a fi corect pentru ele. În acest moment singura informație pe care un bit-node c_i o are, este cea a bit-ului recepționat, y_i. Aceasta inseamna, de exemplu, că c_1 trimite un mesaj continâd “1” catre f_2 și f_4, nodul c_2 trimite mesajul continand $y_2 = 1$ catre f_1 și f_2, și așa mai departe.
2. În pasul al doilea fiecare nod f_j calculeaza raspunsul catre fiecare check-node conectat folosind mesajul recepționat anterior. Mesajul de raspuns, în cazul acesta este valoarea (0 sau 1) pe care fiecare check-node considera a fi corecta pentru fiecare bit-node. Decizia este bazata pe informația celorlalte bit-nodes conectate la acest check-node. Acest răspuns este calculate folosind ecuațiile de verificare a parității. Tabelul de mai jos arata aceste operatiuni: check-node-ul f_1 primeste „1” de la c_4, „0” de la c_5, 1 de la c_8 deci decizia pe care o ia pentru bitul c_2 este 0 după calcului modulo 2: 1 + 0 + 1 + ? (***0***) = 0 și trimite mesajul de raspuns înapoi la c_2. Similar, prmieste „1” de la c_2, „1” de la c_4, 1 de la c_8 prin urmare decizia pe care o ia pentru c_5 este 1 după calculul: (1 + 1 + 1 + ? (***1***) = 0 și trmite mesajul de raspuns pentru bitul c_5.

   Daca în acest moment toate ecuațiile de la nodurile de paritate sunt sătifacute, insemnand că valorile calculate de nodurile de verificare a parității se potrivesc cu valorile primite, algoritmul se termină, altfel se trece la pasul 3.

check nodes	activities				
f_1	receive	$c_2 \rightarrow 1$	$c_4 \rightarrow 1$	$c_5 \rightarrow 0$	$c_8 \rightarrow 1$
	send	$0 \rightarrow c_2$	$0 \rightarrow c_4$	$1 \rightarrow c_5$	$0 \rightarrow c_8$
f_2	receive	$c_1 \rightarrow 1$	$c_2 \rightarrow 1$	$c_3 \rightarrow 0$	$c_6 \rightarrow 1$
	send	$0 \rightarrow c_1$	$0 \rightarrow c_2$	$1 \rightarrow c_3$	$0 \rightarrow c_6$
f_3	receive	$c_3 \rightarrow 0$	$c_6 \rightarrow 1$	$c_7 \rightarrow 0$	$c_8 \rightarrow 1$
	send	$0 \rightarrow c_3$	$1 \rightarrow c_6$	$0 \rightarrow c_7$	$1 \rightarrow c_8$
f_4	receive	$c_1 \rightarrow 1$	$c_4 \rightarrow 1$	$c_5 \rightarrow 0$	$c_7 \rightarrow 0$
	send	$1 \rightarrow c_1$	$1 \rightarrow c_4$	$0 \rightarrow c_5$	$0 \rightarrow c_7$

Figura 2.9 Tabelul 1

3. În urmatoare fază: bit-node –urile c_i primesc informația de la check-nodes și folosesc informația aditională pentru a decide dacă mesajul original este bun printr-o cale simpla în exemplul nostru, votul majoritatii. În cazul nostru, fiecare bit-node are trei surse de informație pentru fiecare bit. Mesajul primit inițial și cele două sugestii primite de la check-nodes. Operația este ilustrată în tabelul de mai jos

message nodes	y_i	messages from check nodes		decision
c_1	1	$f_2 \rightarrow 0$	$f_4 \rightarrow 1$	1
c_2	1	$f_1 \rightarrow 0$	$f_2 \rightarrow 0$	0
c_3	0	$f_2 \rightarrow 1$	$f_3 \rightarrow 0$	0
c_4	1	$f_1 \rightarrow 0$	$f_4 \rightarrow 1$	1
c_5	0	$f_1 \rightarrow 1$	$f_4 \rightarrow 0$	0
c_6	1	$f_2 \rightarrow 0$	$f_3 \rightarrow 1$	1
c_7	0	$f_3 \rightarrow 0$	$f_4 \rightarrow 0$	0
c_8	1	$f_1 \rightarrow 1$	$f_3 \rightarrow 1$	1

Figura 2.10 Tabelul 1

4. Se repetă pasul 2 pană când algoritmul se termina la pasul 2 sau un numar predefinit de iterații.
   În acest exemplu, algoritmul se termina imediat după prima iterație, pentru că ecuațiile sunt satisfăcute, respectiv bitul c_2 este corectat la *0*. [16]

2.5.4 Soft Decision – decodarea sumă - produs

Algoritmul de decodare sumă – produs este un algoritm de decodare soft decision bazat pe transmiterea mesajelor. Este similar cu algoritmul descris în sectiunea anterioara, dar meseajele trimise pe muchiile grafului lui Tanner sunt de aceasta dată probabilități.

Probabilitatățile de la intrare se numesc probabilități *a prioi* pentru că sunt cunoscute inante de a face decodarea LDPC. Probabilitatile returnate de catre decodor se numesc probabilitati *a posteriori*. În acest caz al decodarii sumă produs probabilittile sunt expirimate că raporturi logaritmice.

Pentru o valoare binara *x*, fiind data probabilitatea că *x = 0*, *p(x = 0)*, este usor să aflam probabilitea pentru că *x = 1*, *p(x = 1) = 1 – p(x = 0)*, deci avem nevoie să tinem minte doar o singura valoarea pentru probabilitatea lui *x*. Definim în continuare *L(x)* că logaritm din raportul probabilităților pentru o valoare binară:

$$L(x) = \log\left(\frac{p(x=0)}{p(x=1)}\right) \tag{1.21}$$

unde *log* inseamna log_e. Daca *p(x = 0) > p(x = 1)* atunci *L(x)* este positiv și cu cât diferenta dintre *p(x = 0)* și *p(x = 1)* este mai mare cu atat suntem mai siguri că *p(x) = 0*. Respectiv, dacă *p(x = 1) > p(x = 0)* atunci *L(x)* este negativ iar cu cât diferenta intre *p(x = 1)* și *p(x = 0)* este mai mare cu atat suntem mai sigur că *p(x) = 1*. În consecința semnul lui *L(x)* ne arata decizia asupra lui *x*, iar valoarea absoluta *|L(x)|* ne arata gradul de incredere asupra deciziei. Pentru a avea probabilitati folosim urmatoarea formula:

$$p(x=1) = \frac{p(x=1)/p(x=0)}{1+p(x=1)/p(x=0)} = \frac{e^{-L(x)}}{1+e^{-L(x)}} \tag{1.21}$$

și

$$p(x=0) = \frac{p(x=0)/p(x=1)}{1+p(x=0)/p(x=1)} = \frac{e^{L(x)}}{1+e^{L(x)}}. \tag{1.21}$$

Scopul algoritmului sumă- produs este de a calcula probabilitățile a posteriori maxime - *maximum a posteriori probability* (MAP) pentru fiecare bit al cuvantului de cod, *Pi = P{ci = 1|N}*, care reprezintă probabilitatea că bit-ul *i*din cuvantul de cod este „1” conditionat de evenimentul *N* că toate ecuațiile de verificare a parității sunt satisfăcute. Informația extra despre bitul *i* resceptionată de la ecuațiile de verificare a paritasii se numește informație extrinsecă pentru bit-ul *i.*

Să cosiderăm probabilitatea de a fi un numar par de „1” de la două bit-nodes. Fie q_1 probabilitatea că bitul c_1 să fie 1, respectiv q_2 probabilitatea că bitul c_2 să fie 1. Avem:

$$\begin{aligned} P[c_1 \oplus c_2 = 0] &= q_1q_2 + (1-q_1)(1-q_2) \\ &= 1 - q_1 - q_2 + q_1q_2 \\ &= 1/2(2 - 2q_1 - 2q_2 + 4q_1q_2) \\ &= 1/2[1 + (1-2q_1)(1-2q_2)] = q \end{aligned}$$

În general:

$$\Pr[c_1 \oplus \ldots \oplus c_n = 0] = \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{n}(1-2q_i)$$

$$= P_{j.i}^{ext} \tag{1.22}$$

Numim informația extriseca de la check-node-ul *j* catre bit node *i*, $P_{j.i}^{ext}$, iar $E_{j,i}$ este LLR *(log-likelihood ratio)* al probabilitatii că bit-ul *i* să sătisfaca ecuația de verificare a parității *j*.

Prin urmare, mesajul pe care il trimite check-node-ul f_j catre bit-node-ul c_1 la iteratia *l* este:

$$r_{ji}^{(l)}(0) = \frac{1}{2} + \frac{1}{2}\prod_{i' \in V_j, i' \neq i}(1-2q_{i'j}^{(l-1)}(1))$$

$$r_{ji}^{(l)}(1) = 1 - r_{ji}^{(l)}(0) \tag{1.23}$$

unde V_j este setul de bit-nodes conecate la check-node-ul f_j.

Mesajul pe care un bit-node c_i il trimite catre check-node-ul f_j la iteratia la iteratia *l* este:

$$q_{ij}^{(l)}(0) = k_{ij}(1-P_i)\prod_{j' \in C_i, j' \neq j} r_{j'i}^{(l-1)}(0)$$

$$q_{ij}^{(l)}(1) = k_{ij}P_i\prod_{j' \in C_i, j' \neq j} r_{j'i}^{(l-1)}(1) \tag{1.24}$$

unde C_i este setul de noduri check-node conecate la bit-node-ul c_i.

Constanta k_{ij} este aleasă astfel incat

$q_{ij}(0) + q_{ij}(1) = 1$ [referinta a tutorial]

Exprimat sub forma logaritmica

$$E_{j,i} = \mathrm{LLR}(P_{j,i}^{\mathrm{ext}}) = \log\left(\frac{1-P_{j,i}^{\mathrm{ext}}}{P_{j,i}^{\mathrm{ext}}}\right), \tag{1.25}$$

iar înlocuind $P_{j.i}^{ext}$:

$$E_{j,i} = \log\left(\frac{\frac{1}{2}+\frac{1}{2}\prod_{i' \in Vj}(1-2q_{i'j}^{(l-1)})}{\frac{1}{2}-\frac{1}{2}\prod_{i' \in Vj}(1-2q_{i'j}^{(l-1)})}\right) \tag{1.26}$$

iar folosind relația

$$\tanh\left(\frac{1}{2}\log\left(\frac{1-p}{p}\right)\right) = 1 - 2p \tag{1.27}$$

rezultă

$$E_{j,i} = \log\left(\frac{\frac{1}{2}+\frac{1}{2}\prod_{i' \in Vj}\tanh(M_{j,i'}/2)}{\frac{1}{2}-\frac{1}{2}\prod_{i' \in Vj}\tanh(M_{j,i'}/2)}\right) \tag{1.28}$$

unde

$$M_{j,i'} = LLR\left(P_{j.i}^{int}\right) = \log\left(\frac{1-P_{j.i}^{int}}{P_{j.i}^{int}}\right), \text{ iar } P_{j.i}^{int} = q_{i'j}^{l} \quad (1.29)$$

Fiecare bit are acces la probabilitaile a priori LLR, r_i, și de LLR-urile de la fiecare check-node conectat. Totalul probabilitatilor LLR pentru bitul *i* il reprezintă suma acestor LLR-uri:

$$L_i = LLR\,(P_{j.i}^{int}) = r_i + \sum E_{j,i}.$$

prin urmare

$$M_{j,i} = \sum E_{j',i} + r_i \text{ cu } j' \neq j.$$

Probabilitatile de intrare ale algoritmului sunt

$$r_i = \log\left(\frac{p(c_t=0)}{p(c_t=1)}\right) \quad (1.30)$$

Exemplul 2.8

Cosideram matricea de paritate de mai jos cu ecuațiile de verificare a parității aferente:

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

$c_1 \oplus c_2 \oplus c_4 = 0$

$c_2 \oplus c_3 \oplus c_5 = 0$

$c_1 \oplus c_5 \oplus c_6 = 0$

$c_3 \oplus c_4 \oplus c_6 = 0$

Fie cuvantul de cod

c = [0 0 1 0 1 1],

trimis pe un canal cu o probabilitate de eroare (de schimbare a bitului 0 în 1, sau invers), de *p= 0.2*,

y = [1 0 1 0 1 1]

este recepționat. Pentru că avem un canal pe care se trimit doar două simboluri, probabilitatea că 0 să fi fost trimis, dacă s-a recepționat 1 este *p* (adică a aparut o eroare), iar probabilitatea ca 1 să fi fost trimis dacă s-a recepționat 1 este *1 – p*. Similar, probabilitatea că 1 să fi fost trmis, dacă s-a recepționat 0 este egala cu probabilitatea de eroare *p*, iar probabilitatea că 0 să fi fost trimis, dacă s-a recepționat 0 este egala cu *1 – p* adică probabilitatea de trasmisie corecta. Prin urmare probabilitățile a priori sunt:

$$r_i = \begin{cases} \log\dfrac{p}{1-p}, daca\ y_i = 1 \\ \log\dfrac{p-1}{p}, daca\ y_i = 0 \end{cases}$$

Pentru acest canl avem

$$\log\frac{p}{1-p} = \log\frac{0.2}{0.8} = -1.3863,$$

$$\log\frac{p-1}{p} = \log\frac{0.2}{0.8} = 1.3863,$$

deci probabilitățile a apriori sunt:

$$r = [-1.3863, 1.3863, -1.3863, 1.3863, -1.3863, -1.3863].$$

Setam numarul maxim de iteratii și inițializam $M_{j,i} = r_i$.

Primul bit este inclus în ecuațiile 1 și 3 deci $M_{1,1}$ și $M_{3,1}$ sunt intializate cu r_1:

$$M_{1,1} = r_1 = -1.3863 \text{ și } M_{3,1} = r_1 = -1.3863.$$

Repeteam pentru biții ramasi după aceeasi regula:

i = 2: $M_{1,2} = r_2 = 1.3863$ $M_{2,2} = r_2 = 1.3863$

i = 3: $M_{2,3} = r_3 = -1.3863$ $M_{4,3} = r_3 = -1.3863$

i = 4: $M_{1,4} = r_4 = 1.3863$ $M_{4,4} = r_4 = 1.3863$

i = 5: $M_{2,5} = r_5 = -1.3863$ $M_{3,5} = r_5 = -1.3863$

i = 6: $M_{3,6} = r_6 = -1.3863$ $M_{4,6} = r_6 = -1.3863$

Pasul 1: se calculeaza informația extrinsecă. Prima ecuație de paritate include bitul 1, 2 și 4, deci informația extrinseca pentru bit-ul 1 din prima ecuație de paritate depide de probabilitățile biților 2 și 4:

$$\begin{aligned} E_{1,1} &= \log\left(\frac{1+\tanh(M_{1,2}/2)\tanh(M_{1,4}/2)}{1-\tanh(M_{1,2}/2)\tanh(M_{1,4}/2)}\right) \\ &= \log\left(\frac{1+\tanh(1.3863/2)\tanh(1.3863/2)}{1-\tanh(1.3863/2)\tanh(1.3863/2)}\right) \\ &= \log\left(\frac{1+0.6*0.6}{1-0.6*0.6}\right) = 0.7538. \end{aligned}$$

Similar, informația extrinsecă pentru bit-ul 2 din prima ecuație de paritate depinde de proabilitațile biților 2 și 4:

$$\begin{aligned} E_{1,2} &= \log\left(\frac{1+\tanh(M_{1,1}/2)\tanh(M_{1,4}/2)}{1-\tanh(M_{1,1}/2)\tanh(M_{1,4}/2)}\right) \\ &= \log\left(\frac{1+\tanh(-1.3863/2)\tanh(1.3863/2)}{1-\tanh(-1.3863/2)\tanh(1.3863/2)}\right) \\ &= \log\left(\frac{1+-0.6*0.6}{1--0.6*0.6}\right) = -0.7538, \end{aligned}$$

și pentru bit-ul 4 din prima ecuație de verificare a parității, informația extrinsecă depinde de probabilitățile biților 1 și 2:

$$\begin{aligned} E_{1,4} &= \log\left(\frac{1+\tanh(M_{1,1}/2)\tanh(M_{1,2}/2)}{1-\tanh(M_{1,1}/2)\tanh(M_{1,2}/2)}\right) \\ &= \log\left(\frac{1+\tanh(-1.3863/2)\tanh(1.3863/2)}{1-\tanh(-1.3863/2)\tanh(1.3863/2)}\right) \\ &= \log\left(\frac{1+-0.6*0.6}{1--0.6*0.6}\right) = -0.7538. \end{aligned}$$

În continuare, a două ecuație de verificare a parității include biții 2 3 și 5 deci informația extrinsecă calculata este:

$$\begin{aligned}
E_{2,2} &= \log\left(\frac{1+\tanh(M_{2,3}/2)\tanh(M_{2,5}/2)}{1-\tanh(M_{2,3}/2)\tanh(M_{2,5}/2)}\right) \\
&= \log\left(\frac{1+\tanh(-1.3863/2)\tanh(-1.3863/2)}{1-\tanh(-1.3863/2)\tanh(-1.3863/2)}\right) \\
&= \log\left(\frac{1+-0.6*-0.6}{1--0.6*-0.6}\right) = 0.7538, \\
E_{2,3} &= \log\left(\frac{1+\tanh(M_{2,2}/2)\tanh(M_{2,5}/2)}{1-\tanh(M_{2,2}/2)\tanh(M_{2,5}/2)}\right) \\
&= \log\left(\frac{1+\tanh(+1.3863/2)\tanh(-1.3863/2)}{1-\tanh(+1.3863/2)\tanh(-1.3863/2)}\right) \\
&= \log\left(\frac{1+0.6*-0.6}{1-0.6*-0.6}\right) = -0.7538, \\
E_{2,5} &= \log\left(\frac{1+\tanh(M_{2,2}/2)\tanh(M_{2,3}/2)}{1-\tanh(M_{2,2}/2)\tanh(M_{2,3}/2)}\right) \\
&= \log\left(\frac{1+\tanh(+1.3863/2)\tanh(-1.3863/2)}{1-\tanh(+1.3863/2)\tanh(-1.3863/2)}\right) \\
&= \log\left(\frac{1+0.6*-0.6}{1-0.6*-0.6}\right) = -0.7538.
\end{aligned}$$

Repetând procedeul pentru toate check-node-urile obținem:

$$E = \begin{bmatrix} 0.7538 & -0.7538 & . & -0.7538 & . & . \\ . & 0.7538 & -0.7538 & . & -0.7538 & . \\ 0.7538 & . & . & . & 0.7538 & 0.7538 \\ . & . & -0.7538 & 0.7538 & . & -0.7538 \end{bmatrix}.$$

În continuare calculăm probabilitățile pentru fiecare bit. Bit-ul 1 are imformatiile extrinseci de la check-node-urile 1 și 3 și informația intrinsecă primita pe canal. LLR-ul total pentru un bit il reprezintă suma lor:

$$L_1 = r_1 + E_{1,1} + E_{3,1} = -1.3863 + 0.7538 + 0.7538 = 0.1213.$$

Se vede că desi LLR-ul prmit pe canal este negativ, ceea ce inseamna că bitul este 1, informatiile extrinseci sunt ambele pozitive, ceea ce inseamna că bitul este 0. Informatiile extrinseci sunt suficient de „puternice" pentru că per total decizia asupra valorii bit-ului să se schimbe efecriv. Procedeul se repeta și pentru restul biților:

$$L_2 = r_2 + E_{1,2} + E_{2,2} = 1.3863$$

$$L_3 = r_3 + E_{2,3} + E_{4,3} = -2.8938$$

$$L_4 = r_4 + E_{1,4} + E_{4,4} = 1.3863$$

$$L_5 = r_5 + E_{2,5} + E_{3,5} = -1.3863$$

$$L_6 = r_6 + E_{3,6} + E_{4,6} = -1.3863$$

Decizia hard asupra biților receptionati este data semnul LLR-urilor calculate mai sus,

$$z = [0\ 0\ 1\ 0\ 1\ 1].$$

Pentru a verifica dacă **z** cuvânt de cod valid

$$\mathbf{s} = \mathbf{z}H' = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}.$$

Se observa că z este un cuvânt de cod valid, deci decodarea se opreste și se returneaza **z** că un și cuvantul decodat [2].

III. Aplicație de simulare coduri LDPC, CAMAG v7.2

3.1 Prezentare simulator CAMAG 6

Programul de simulare a canalelor magnetice realizat permite simularea mai multor tipuri de canale: clasic, PR4, EPR4 și E2PR4. Programul realizează simularea începând de la precodarea și până la detecţia semnalului și dezvoltă simulatorul prezentat în [15]. Schema bloc de realizare a programului este prezentată în figura 13.

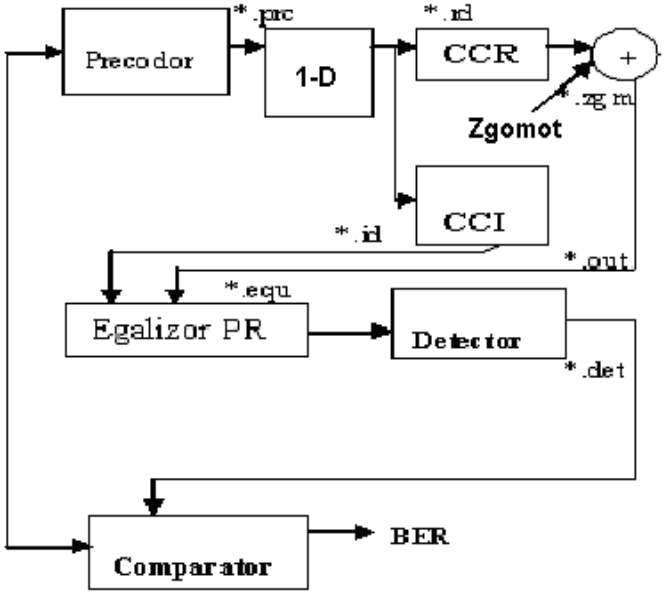


Figura 3.1: Schema bloc a simulatorului

Primul bloc realizează precodarea în scopul reducerii propagării erorilor ce ar putea avea loc la detecţie. Formulele pentru precodare sunt următoarele: pentru canalul clasic: (1+D) mod2; pentru canalul PR4: $(1+D^2)$ mod2; pentru canalul EPR4 $(1-D+D^2+D^3)$ mod2; pentru canalul E^2PR4 $(1-2D+2D^3+D^4)$ mod2.

Blocul **1-D** realizează o simulare a caracteristicii diferenţiale a capului de citire.

Blocul **CCR** (caracteristică canal real) aproximează caracteristica canalului folosind caracteristica Lorentziană. Egalizorul utilizează pentru antrenare un algoritm LMS ce foloseste un număr de pasi și o rată de antrenare stabilite de utilizator.

Blocul **CCI** (caracteristică canal ideal) simulează situaţia în care canalul ar fi lipsit de zgomot. În cazul în care se lucrează cu secvenţe codate RLL, codarea este realizată tot de precodor, fisierul rezultat având extensia **.rll,** iar decodarea RLL este efectuată de detector, evident după detecția propriu-zisă, rezultând un fisier cu extensia **.dec**. Detecția poate fi realizată folosind două tipuri de detectoare: detectorul de prag și detectorul Viterbi. Rezultatul simulării il reprezintă calculul ratei de erori **BER** (Bit Error Rate).

Interfața:

Figura 3.2: Interfața CAMAG 6

În urma apăsării butonului de Opțiuni va aparea fereastra de dialog:

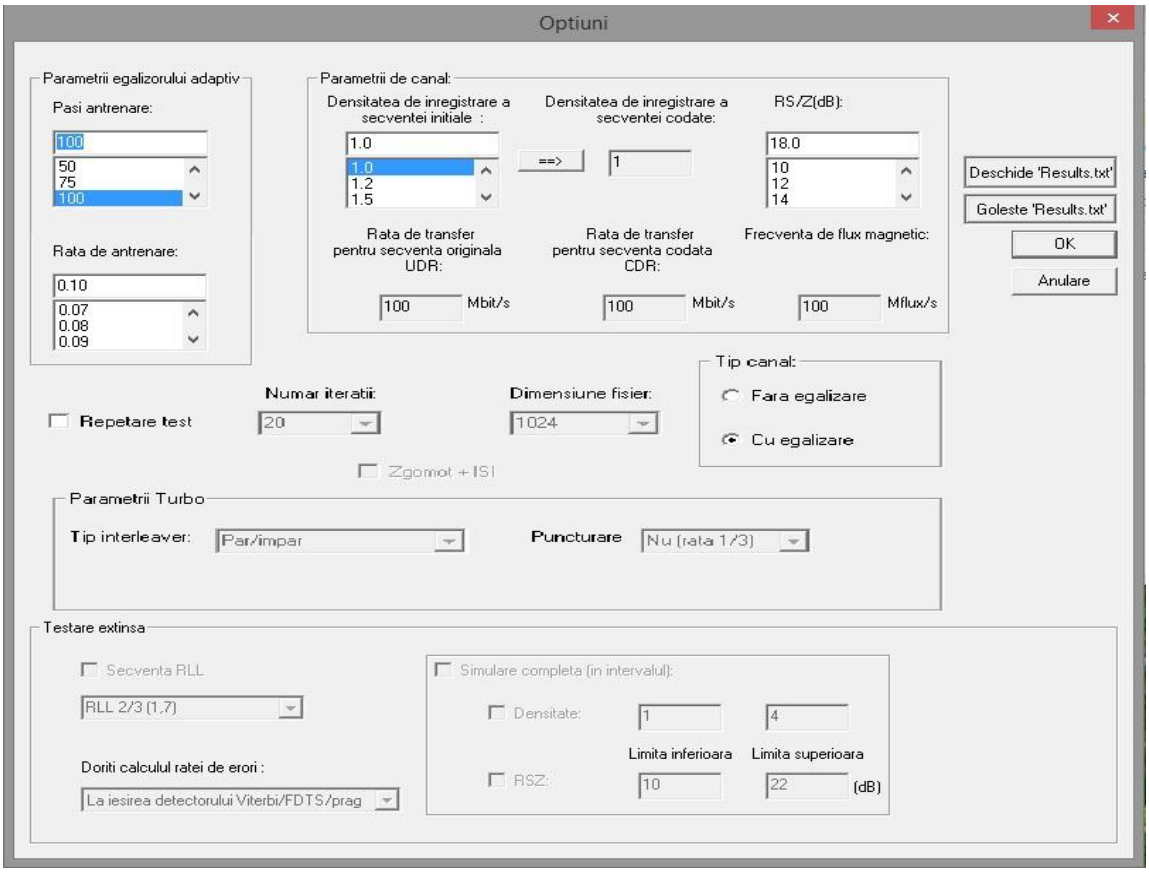


Figura 3.3 Fereastra Optiuni din CAMAG 6

Aici se pot seta diferiti parametri pentru codarea și decodarea TURBO.

3.2 Îmbunătățiri aduse de CAMAG 7.2

În versiunea CAMAG 7.2 s-a introdus modulul de codare - decodare pentru codurile LDPC. Atat codarea cât și decodarea se face după principiul algoritmilor prezentati în capitolul anterior. După cum se vede și în figura 3.4, am schimbat interfata grafica a simulatorului adaugând modului de LDPC, un buton separat pentru a selecta optiunile pentru codarea – decodarea LDPC și un buton pentru a incepe simularea LDPC.

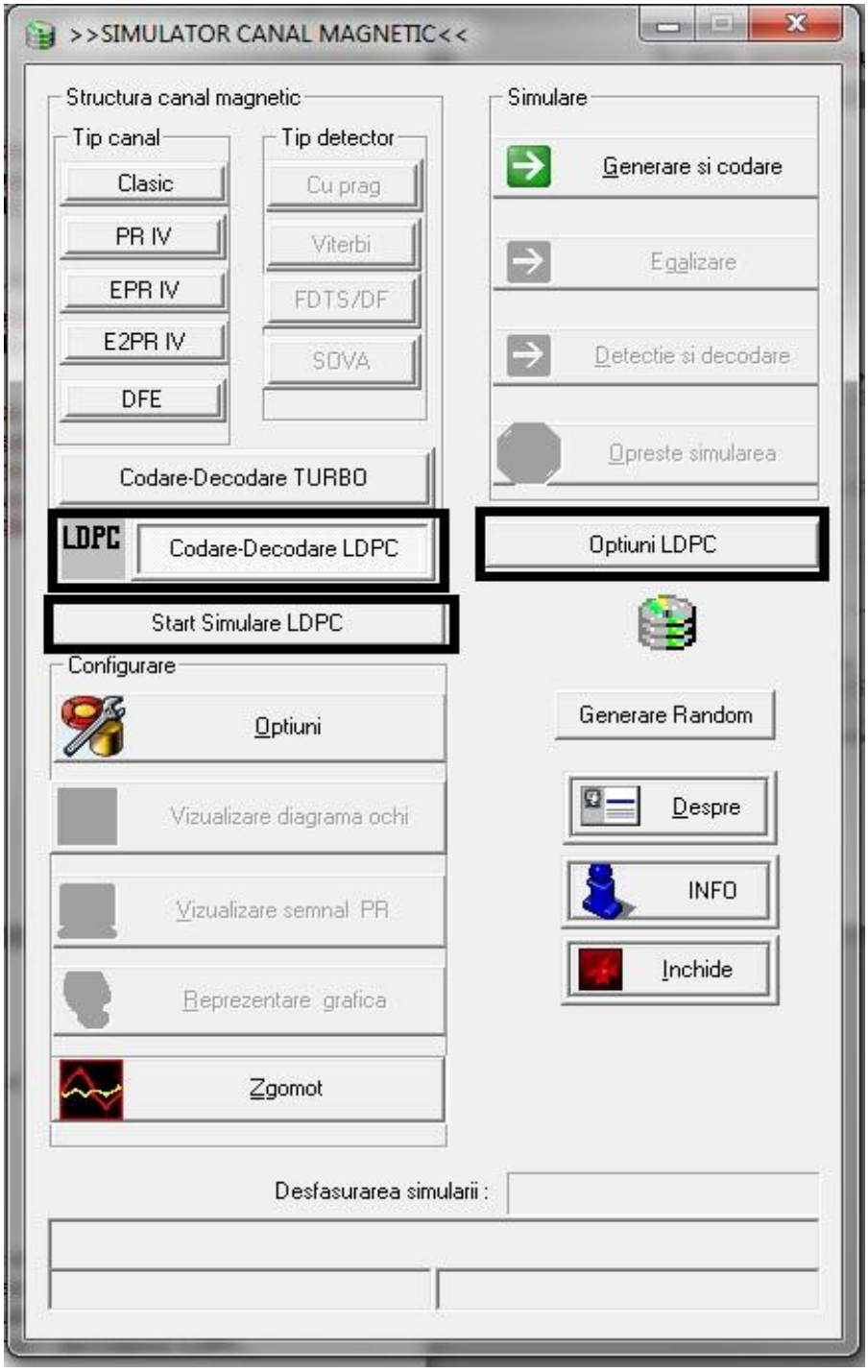


Figura 3.4 Interfața CAMAG 7.2

Prin apasărea butonului 

se activează modulul LDPC impreuna cu butoanele aferente. Cand acesta nu este apasăt se activeaza modulul de decodare TURBO că în figura de mai jos:

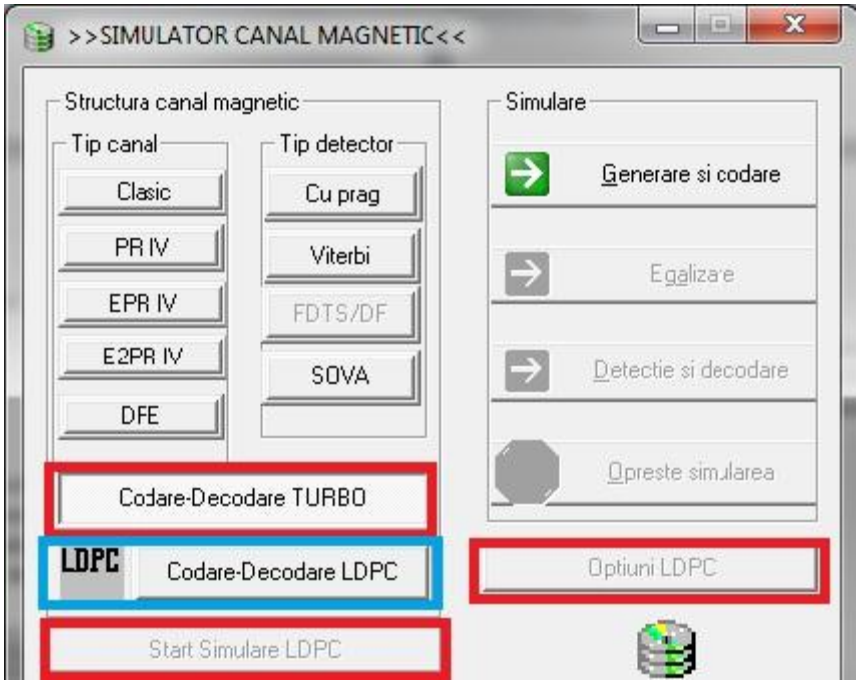


Figura 3.5

Butonul 

deschide o noua fereastră unde se pot seta parametri pentru modulul de codare – decodare LDPC. Parametri care se pot seta sunt M = numarul de rânduri, N = numarul de coloane, numarul de valori diferite de 0 de pe coloană, numarul de iteratii și lunigimea mesajului. Rata mesajului trebuie să fie de $\frac{1}{2}$ adică $\frac{M}{N} = \frac{1}{2}$. Mesajul se codeaza în blocuri de cate M biți. Fereastra pentru configurarea optiunilor arată că în figura de mai jos:

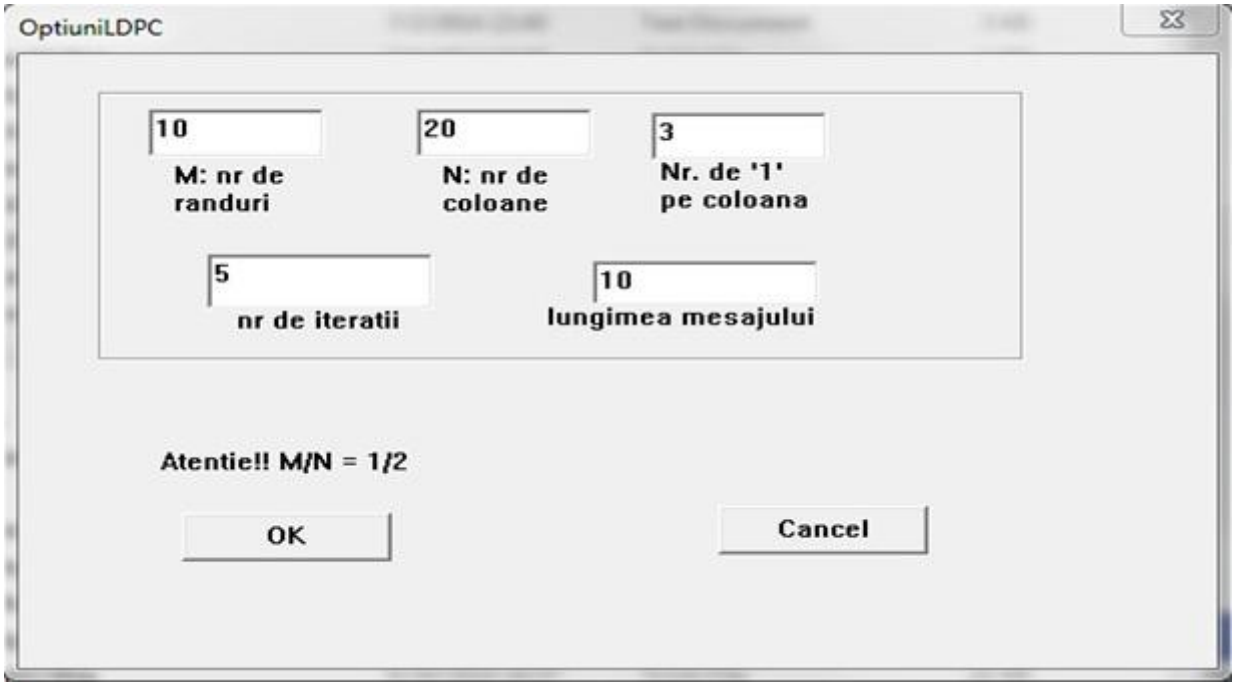


Figura 3.6 Fereastra OptinuiLDPC din CAMAG 7.2

Butonul 

porneste simularea de codare și decodare LDPC. Se deschide o fereastra noua în care se deseneaza un grafic, valorile BER – ului se scriu atat în fereastra Simulare LDPC cât și în fisierul *ldpcBER_log.txt.* În figura de mai jos este prezentată fereastra Simulare LDPC.

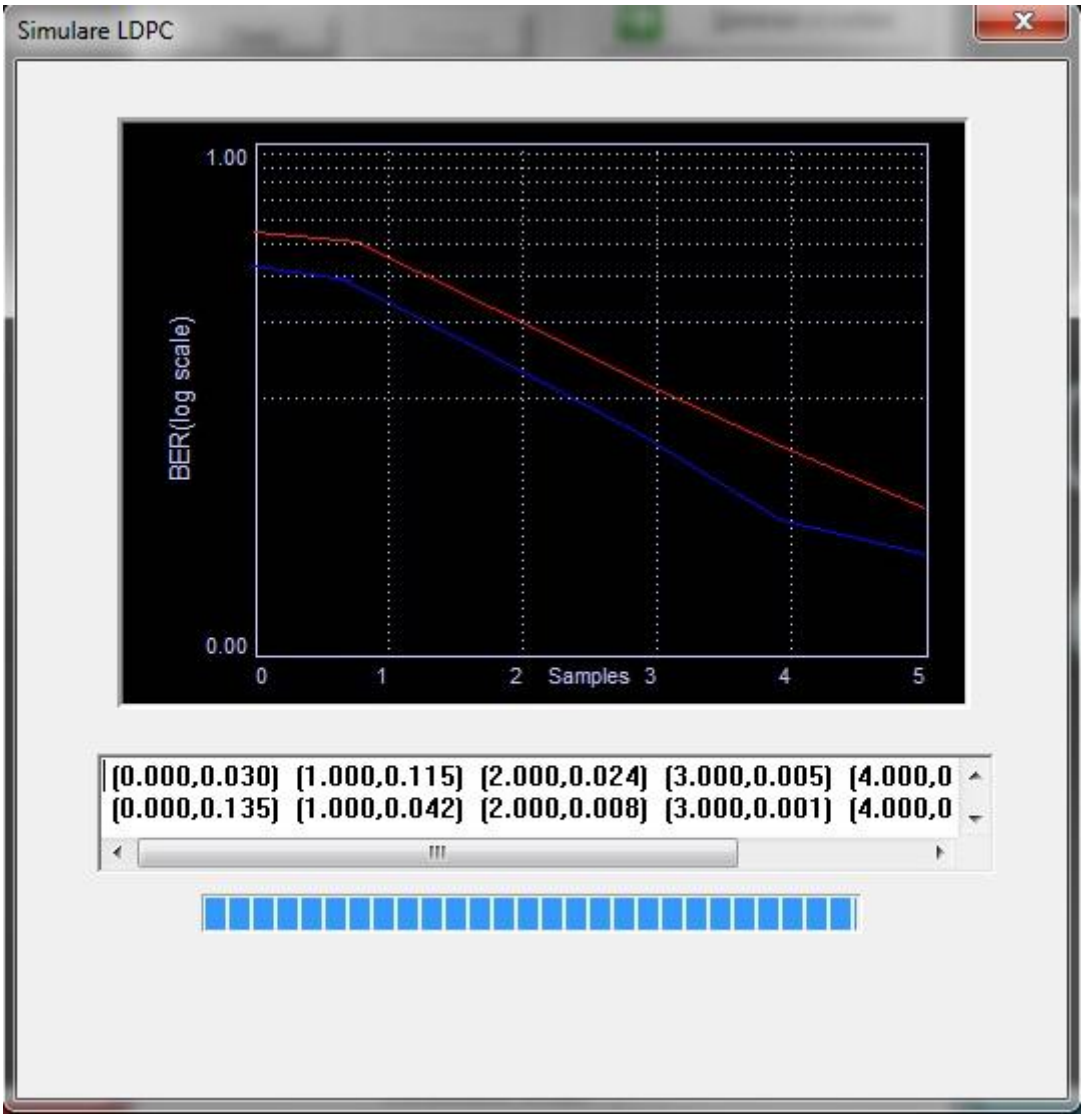


Figura 3.7 Fereastra Simulare LDPC

IV. Rezultate experimentale și analiza rezultatelor

Simularile s-au efectuat cu algoritmii prezentati în capitolul 2 și anume, algorimul hard – decision, Bit – Flipping și algoritmul soft – decision, Sumă – Produs.

Am testat codurile în mai multe situatii pentru a evidenția performatele lor.

1. Inițial coparăm rezultatele obținunte în urma simularilor pe matrici de decodare nu foarte mari, unde lungimea mesajlui este constantă, dar variem numarul de iteratii.

În figura 4.1 de mai jos este rezultatul simulării efectuate cu o matrice de verificare a parității de dimensiunile 256 x 512, cu o lungime a mesajului de 1024 de biți, testul repetânduse de **5 ori**

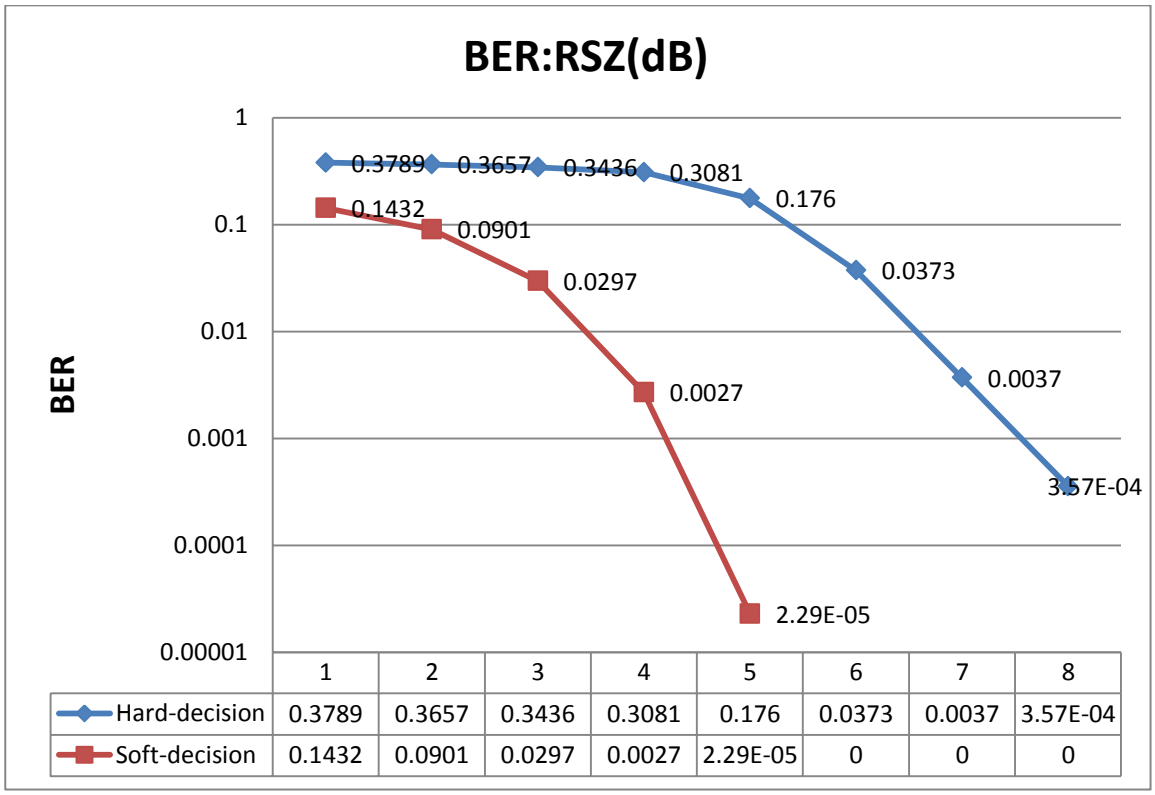


Figura 4.1 BER in funcție de RSZ, iterații variabile

În figura 4.2 de mai jos este rezultatul simulării efectuate cu o matrice de verificare a parității de dimensiunile 256 x 512, cu o lungime a mesajului de 1024 de biți, testul repetânduse de **15 ori** de această dată.

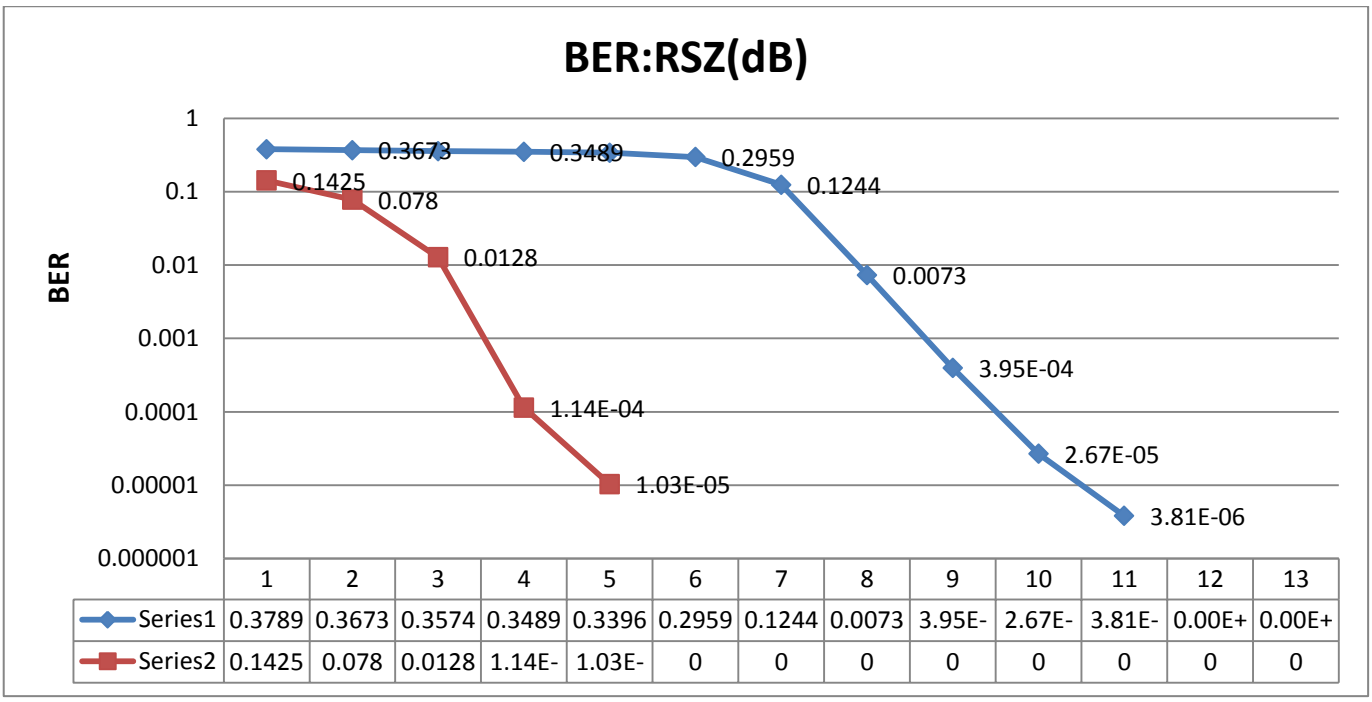


Figura 4.2 BER in funcție de RSZ, iterații variabile

2. În a două situatie se păstrează lungimea mesajului și numarul de iteratii constante și se variaza dimensiunile matricilor.

În figura 4.3 de mai jos este rezultatul simulării efectuate cu o matrice de verificare a parității de dimensiunile **128 x 256**, cu o lungime a mesajului de 1024 de biți, testul repetânduse de 5 ori.

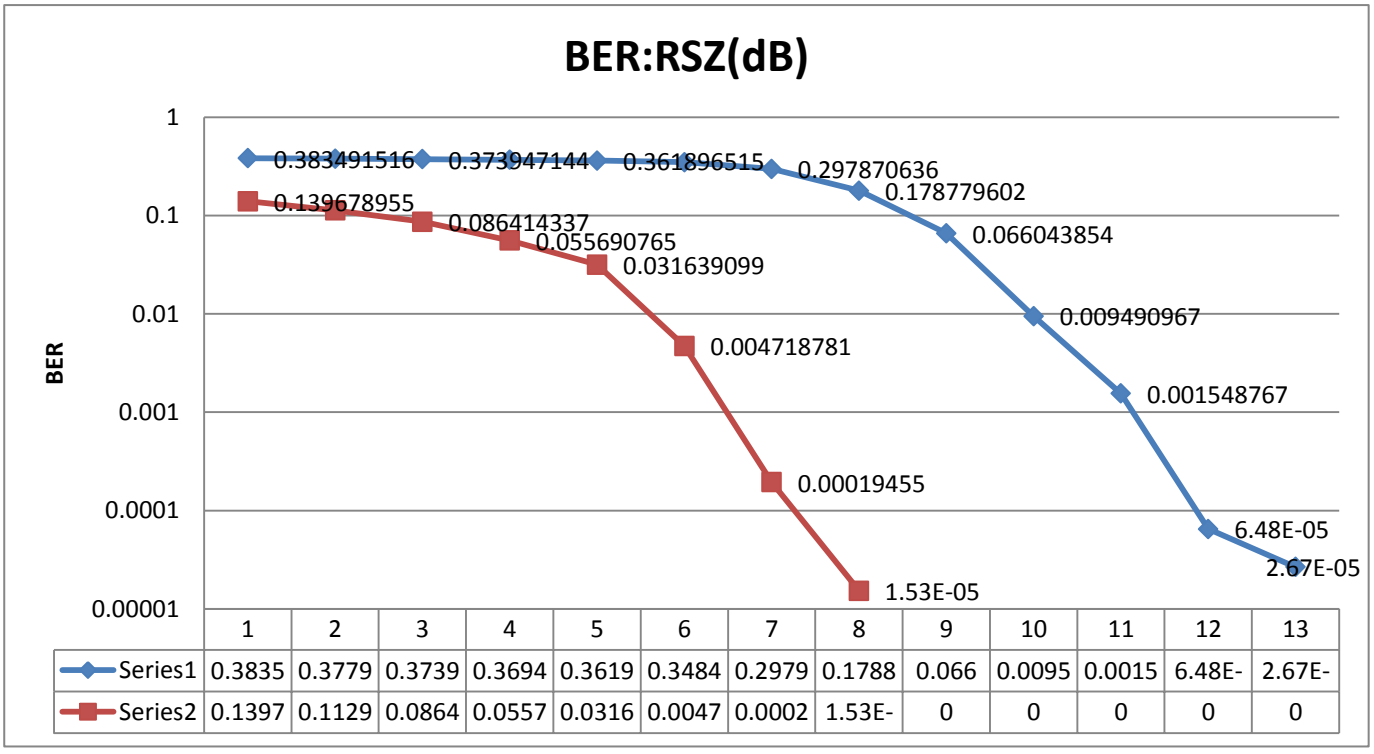


Figura 4.3 BER in funcție de RSZ, dimensiunea matricii variabilă

În figura 4.4 de mai jos este rezultatul simulării efectuate cu o matrice de verificare a parității de dimensiunile **512 x 1024**, cu o lungime a mesajului de 1024 de biți, testul repetânduse de 5 ori.

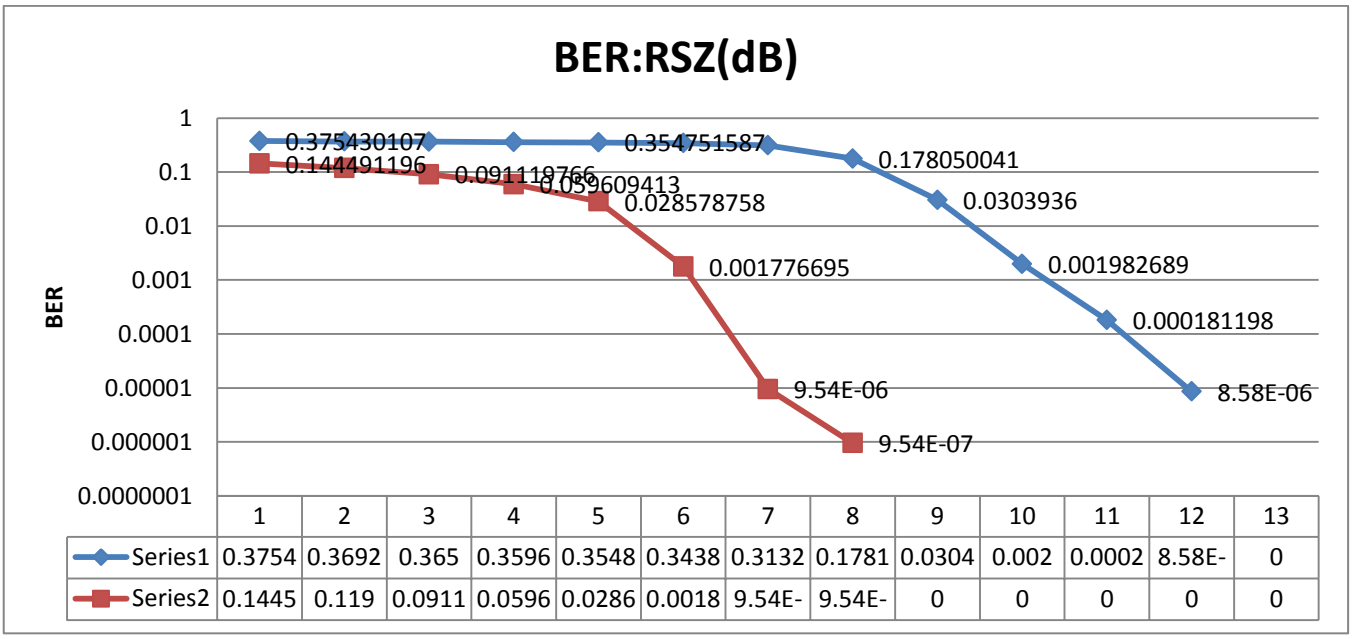


Figura 4.4 BER in funcție de RSZ, dimensiunea matricii variabilă

3. În a treia situație se variază dimensiunea mesajului dar se pastreaza constant numarul de iteratii și dimensiunea matricilor.

În figura 4.5 de mai jos este rezultatul simulării efectuate cu o matrice de verificare a parității de dimensiunile 128 x 256, cu o lungime a mesajului de **1024** de biți, testul repetânduse de 5 ori.

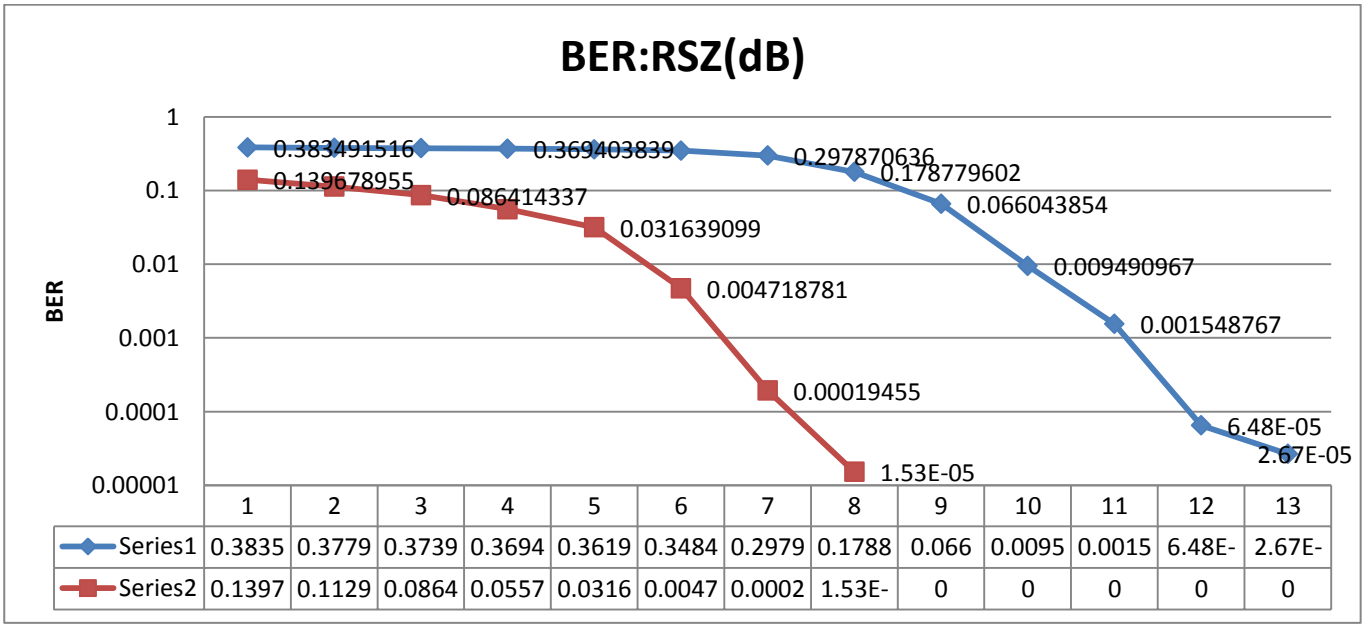


Figura 4.5 BER in funcție de RSZ, lungimea mesajului variabilă

În figura 4.6 de mai jos este rezultatul simulării efectuate cu o matrice de verificare a parității de dimensiunile 128 x 256, cu o lungime a mesajului de **10240** de biți, testul repetânduse de 5 ori.

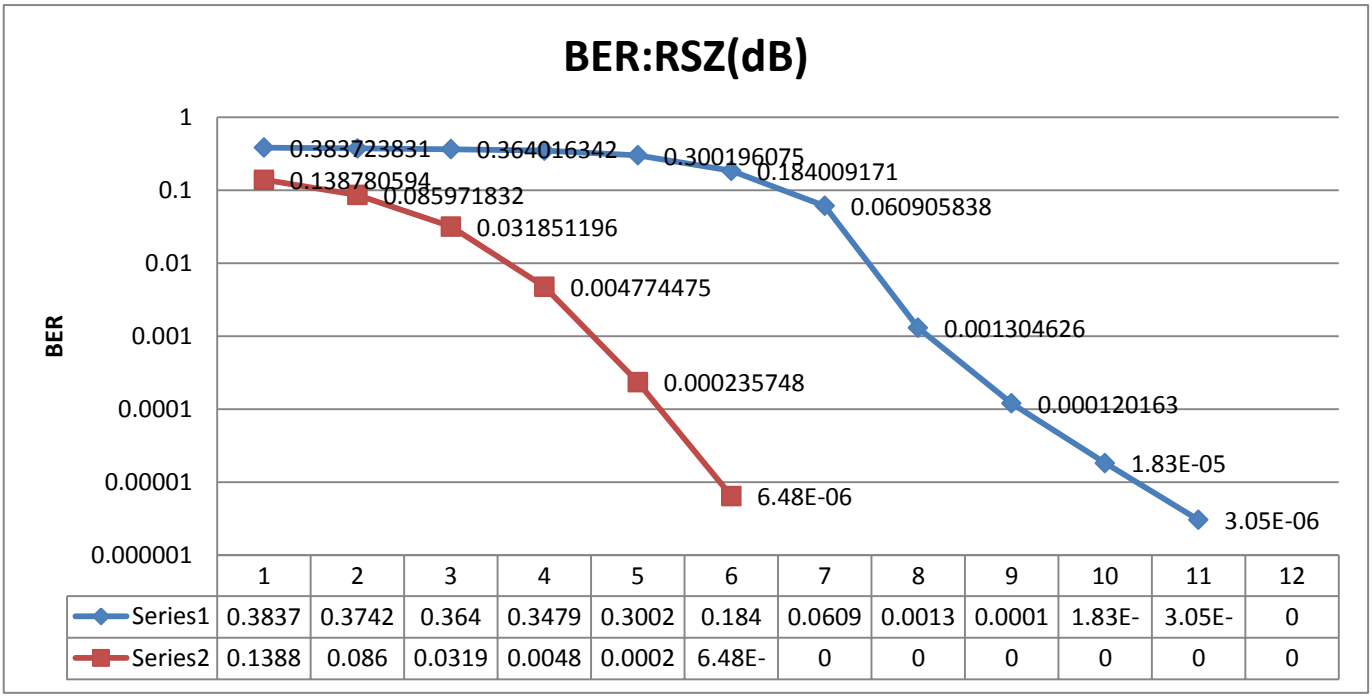


Figura 4.6 BER in funcție de RSZ, lungimea mesajului variabilă

Se observă că in cazul primului set de simulări peroformanța obținută cu 15 iteratii este mai bună decât in cazul repetarii testului de doar 5 ori. În cazul algoritmului hard –decision performanțele sunt aproximativ aceleași in amblele cazuri, pe când algoritmul soft – decision obține performanțe mai bune, lucru vizibil prin modul in care descreşte rata de eroare: în cazul in care am efectuat doar 5 iterații observăm ca o rată de erori acceptabilă se obține incepand cu un raport semnal zgomot de 5 dB. În cazul in care s-au efectuat 15 iterații se observă că raportul semnal zgomot de la care algoritmul de decodare se poate folosi cu erori minime scade la 5 la 4 dB.

Pentru al doilea scenariu de simulări, în care se variază dimensiunea matricilor se observă o dată cu cresterea matricii se obține o rată de erori mai buna in cazul algoritmului soft – decision și performanțe mai bune un RSZ mai mic. Pentru matricea de dimensiuni 128 x 256 se vede ca putem folosi codul de la un RSZ de 8 dB, iar când matricea crește la dimensiunea de 512 x 1024, se observă ca obținem aleseași erori pentru un RSZ de aproximativ 6.5 dB. Același lucru se observă si pentru algoritmul hard – decision, când în primul caz, eroarea aceeptabilă o obținem de la un RSZ de 12 dB, iar în cazul al doilea RSZ scade la 11 dB si avem un BER mai bun.

Setul al treilea de rezultate ne arata ca o dată cu dimensiunea mesajului mai mare, scade si rata de erori, pentru ambii algoritmi testați. Se vede că cazul al doilea în care mesajul este de 10 ori mai mare decat în primul caz, s-au obținut rate de erori mai bune si un RSZ mai mic pentru ambii algoritmi: BER-ul scade cu un ordin de mărime, iar RSZ cu aproximativ 2 unități.

În general, algoritmul soft – decision este superior algoritmului hard – decison dupa cum ne asteptam. Algoritmul hard decision se foloseste in general pentru simulari in laborator, tutoriale penrtru intelegerea codurilor, deoarece nu implica calcul probabilistic.

III. Concluzii

Această lucrare prezintă conceptele impotante ale codruilor de control al parității cu densitate mică (LDPC): codarea, decodarea, principalii algoritmi folosiți pentru decodare. Codurile LDPC au fost studiate intens în ultimii ani și s-au facut progrese imense, de la introducerea loc de către Gallager în 1963, în întelegerea și abilitatea de a proiecta decodarea iterativă. Rezultatele obtinuțe de cercetatori ca David J. C. MacKay [2] și Jing Li, Krishna R. Narayanan, Costas N. Georghiades [21], arată că aceste coduri se apropie de limita lui Shannon la aproximativ 0.04 dB la un BER de 10^{-6} și o lungime a blocului de date de 10^{7}[17]. Se observă din simulările efectuate că perfomrmanțele algoritmului soft- decision sunt net superioare algorimului hard – decision și comparabile cu cele existente în domeniu la momentul actual. Un dezavantaj al acestor coduri este complexitatea codorului și după cum se observă din simulările efectuate lungimea mesajului trebuie să fie mare, ceea ce implică timpi de decodare foarte mari. O soluție pentru această problemă o reprezintă calculul paralel (pentru mai multe delaii a se vedea [22], iar pentru mai detalii privind implementarea hardwere, care nu s-a abordat în aceasta lucreare, a se vedea [23]).

Referințe

[1] **R. G. Gallager**, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[2] **D. J. C. MacKay**, "*Good error-correcting codes based on very sparse matrices*," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, March 1999.

[3] **Prof. Dr. ing. Inge Gavat** Curs TTI 2003 (sem. vara)

[4] http://www.scientia.ro/stiinta-la-minut/119-concepte-explicate-in-60-de-secunde/3144-ce-este-limita-shannon.html accesat la 03.06.2014 17:58

[5] http://mathworld.wolfram.com/Error-CorrectingCode.html accesat la 03.06.2014 17:58

[6] http://stud.usv.ro/TTI/LAB/Lab_5_2010_2011.pdf accesat la 03.06.2014 17:58

[7] **R. M. Tanner**, "*A recursive approach to low complexity codes," IEEE Trans, In- form. Theory*, vol IT-27, pp. 533-547, Sept. 1981.

[8] **Sarah J. Johnson**, *"Introducing Low-Density Parity-Check Codes",* School of Electrical Engineering and Computer Science, The University of Newcastle, Australia

[9] **D. J. C. MacKay,** "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, March 1999.

[10] **D. A. Spielman,** "*Linear-time encodable and decodable error-correcting codes."* IEEE Trans. Inform. Theory, vol. 42, no. 6, pp. 1723 -1731, Nov. 1996.

[11] **M. G. Luby, M. A. Shokrolloahi, M. Mizenmacher, D. A. Spielman**, "Improved low-density parity-check cedes using irregular graphs," IEEE Tmns, Inform, Theory, vol. 47, no, 2, pp. 585-598, Feb. 2001.

[12] **R. Echard și S. C. Chang,** *„The π-rotation low-density parity check code."* In Proc. GLOBECOM 2001, vol. 2, San Antonio, TX, 2001, pp. 980-484

[13] **M. Sipser si D. A. Spielman,** „Expander codes." IEEE Trans, Inform, Theory, ol. 42, no. 6, pp. 1710-1722, Nov. 1996 [14] **T. Zhang si K. K. Parhi**, "*Joint (3,k)-regular LDPC code and decoder/encoder design."* IEEE Trans. Sig. Proc., vol. 52, no. 4, pp. 1065-1079, Apr. 2004.

[15] **St. Stancescu, C. Sandu**, *„Random Interleaver Design in Turbo Coding for Digital Magnetic Recording Channel"*,Information & Communication Technology Electronics & Microelectronics (MIPRO), 2013 36th International Convention, pp. 137-139 , May 2013

[16] **Tuan Ta**, "*A Tutorial on Low Density Parity-Check Codes",* The University of Texas at Austin

[17] **B. M. J. Leiner**, *"LDPC Codes – a brief Tutorial,"* April 2005.

[18] **T. J. Richardson și R. L. Urbanke**, "*Efficient encoding of low-density parity-check codes," IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 638–656, February 2001.

[19] **S. B.Wicker**, „*Error Control Systems for Digital Communication and Storage"*. Upper Saddle River, NJ 07458: Prentice Hall, 1995.

[20] **S. Lin and D. J. Costello Jr**., *Error Control Coding*, 2nd ed. New Jersey: Prentice Hall, 2004.

[21] **Jing Li, Krishna R. Narayanan, Costas N. Georghiades,** *„On the Performance of High-Rate TPC/SPC Codes and LDPC Codes Over Partial Response Channels"*, IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. 50, NO. 5, pp 723 – 734, MAY 2002.

[22]**Esa Alghonaim, Aiman El-Maleh, Adnan Al-Andalusi** *"PARALLEL COMPUTING PLATFORM FOR EVALUATING LDPC CODES PERFORMANCE",* 2007 IEEE International

Conference on Signal Processing and Communications (ICSPC 2007), pp.175-160 24-27 November 2007, Dubai, United Arab Emirates

[23]**Marjan Karkooti, Predrag Radosavljevic, Joseph R. Cavallaro,** *"Configurable, High Throughput, Irregular LDPC Decoder Architecture: Tradeoff Analysis and Implementation,"* Rice Digital Scholarship Archive, September 01, 2006.

Anexa Cod Sursă

******decodeLogDomain.cpp******

```
#include "global.h"

int *decodeLogDomain(float *rx, int **H, int N0,int  iteration,float **Lrji,float **Pibetaij,float
**Lqij,int *r1,int *c1,int *r,int *c,float *Lci,int **alphaij,float **betaij,int *vHat)
{
	int i,j;
	int M = _msize(H)/sizeof(H)-1;
	int N =  _msize(H[1])/sizeof(H[1])-1;
	int k,n,l,t;
	float sumOfPibetaij;
	float prodOfalphaij,PiSumOfPibetaij;
	float LQi;
	int jj;
	for(k = 1;k<=N;k++)
	{
		Lci[k]=(-4*(float)rx[k]/N0);
	}
	Lrji = (float**)zeros(M, N,(int**)Lrji);
	Pibetaij = (float**)zeros(M, N,(int**)Pibetaij);
	Lqij = (float**)zeros(M, N,(int**)Lqij);
	for(k = 1;k<=M;k++)
		for(t = 1;t<=N;t++)
		{
			Lqij[k][t] = (float)H[k][t]*Lci[t];
		}
		jj = 1;
			for(k = 1;k<=N;k++)
				for(j = 1;j<=M;j++)
				  {
					if(H[j][k])
					{
						r[jj] = j;
						c[jj] = k-1+1;
						jj++;
					}

				  }
	for (n = 1;n<=iteration;n++)
	{
	for(k = 1;k<=M;k++)
		for(t = 1;t<=N;t++)
		{
			alphaij[k][t] = Lqij[k][t]>0?1:((Lqij[k][t]==0)?0:-1);
		}
	for(k = 1;k<=M;k++)
		for(t = 1;t<=N;t++)
		{
			betaij[k][t]  = Lqij[k][t]>0?Lqij[k][t]:-1*Lqij[k][t];
```

```
        }

for (l = 1;l<=(jj-1);l++)
{
        Pibetaij[r[l]][c[l]] = log((exp(betaij[r[l]][c[l]])+1)/(exp(betaij[r[l]][c[l]])-1));
}
for (i = 1;i<=M;i++)
{
        jj = 1;

                for(k = 1;k<=N;k++)
                {
                        if(H[i][k]!=0)
                        {
                                c1[jj] = k;
                                jj++;
                        }
                }
        for( k = 1;k<=(jj-1);k++)
        {


                sumOfPibetaij = 0;
                prodOfalphaij = 1;
                for(int hh = 1;hh<=(jj-1);hh++)
                {
                        sumOfPibetaij += (float)Pibetaij[i][(int)c1[hh]];
                }
                sumOfPibetaij -= (float)Pibetaij[i][(int)c1[k]];
                if (sumOfPibetaij < (float)(1e-20))
                        sumOfPibetaij = (float)(1e-10);
                PiSumOfPibetaij = log((exp(sumOfPibetaij) + 1)/(exp(sumOfPibetaij) - 1));

                prodOfalphaij = 1;
                for(int ll = 1;ll<=(jj-1);ll++)
                {
                        prodOfalphaij *= alphaij[i][(int)c1[ll]];

                }
                prodOfalphaij *= alphaij[i][(int)c1[k]];
                Lrji[i][(int)c1[k]] = prodOfalphaij*PiSumOfPibetaij;

        }

}

for (j = 1;j<=N;j++)
{
        jj = 1;
                for(k = 1;k<=M;k++)
                {
                        if(H[k][j]!=0)
```

```
                        {
                                r1[jj] = k;
                                jj++;
                        }
                }
        for (k = 1;k<=(jj-1);k++)
        {
                Lqij[r1[k]][j] = Lci[j] - Lrji[r1[k]][j];
                for(int gg = 1;gg<=(jj-1);gg++)
                {
                        Lqij[r1[k]][j] += Lrji[r1[gg]][j];
                }

        }

        LQi = Lci[j];
        for(int z = 1;z<=(jj-1);z++)
                LQi += (float)Lrji[r1[z]][j];

        if (LQi < 0)
                vHat[j] = 1;
        else
                vHat[j] = 0;
    }

  }

  return vHat;
}
```

****** decodeBitFlip.cpp******

```
#include "global.h"
int *decodeBitFlipping(float *rx, int **H,int iteration)
{


  int M = _msize(H)/sizeof(H)-1;
  int N = _msize(H[1])/sizeof(H[1])-1;

  float *ci = (float*)sign_V(rx);
  for(int hh = 1;hh<=length((float*)rx);hh++)
        ci[hh] = (ci[hh]+1)/2;
  int **rji = new int *[M+1];

  for( int k = 0;k<=M;k++)
  {
        rji[k] = new int [N+1];

  }
```

```
rji = zeros(M, N,rji);

int **qij;
int *c1,*r1;

c1 = new int [N+1];
int **ff;
int  numOfOnes;
int * vHat= new int [M+1];

qij = new int *[M+1];
int k,t,n,i,j,uu,jj;
for(k = 1;k<=M;k++)
{
        qij[k] = new int [N+1];

}
for(k = 1;k<=M;k++)
        for(t = 1;t<=N;t++)
        {

                qij[k][t] = H[k][t]*ci[t];

        }

        float tempf;
for (n = 1;n<=iteration;n++)
{

  for (i = 1;i<=M;i++)
  {

          jj = 1;

                for(k = 1;k<=N;k++)
                {
                        if(H[i][k]!=0)
                        {
                                c1[jj] = k;
                                jj++;
                        }
                }
          for (k = 1;k<=(jj-1);k++)
          {
                  tempf = 0;
                  for(int zz = 1;zz<=(jj-1);zz++)
                  {
                          tempf += qij[i][c1[zz]];
                  }

                  rji[i][c1[k]] = ((int)tempf + (int)qij[i][c1[k]])%2;
                  if(rji[i][c1[k]]<0)
```

```
                                rji[i][c1[k]] = (-1)*rji[i][c1[k]];

                        }

                }

                for (j = 1;j<=N;j++)
                {

                        jj = 1;
                                for(k = 1;k<=M;k++)
                                {
                                        if(H[k][j]!=0)
                                        {
                                                r1[jj] = k;
                                                jj++;
                                        }
                                }

                                ff = new int* [jj-1];
                                for( uu = 1;uu<=(jj-1);uu++)
                                {
                                        ff[uu] =new int [N+1];
                                }

                                for( uu = 1;uu<=(jj-1);uu++)
                                        ff[uu][j] = rji[r1[uu]][j];



                        int  numOfOnes = length((float*)find_Row2(ff,j));
                        for (k = 1;k<=length((float*)r1);k++)
                        {
                                if ((numOfOnes + ci[j]) >= (length((float*)r1) - numOfOnes + rji[r1[k]][j]))
                                        qij[r1[k]][j] = 1;
                                else
                                        qij[r1[k]][j] = 1;
                        }

                        if ((numOfOnes + ci[j]) >= (length((float*)r1) - numOfOnes))
                                vHat[j] = 1;
                        else
                                vHat[j] = 0;

                }

                }
        return vHat;
}
float *sign_V(float *Lqij)
{
        int M = _msize(Lqij)/sizeof(Lqij)-1;
```

```
	float *alphaij;
	int t;
	alphaij = new float [M+1];

	for(t = 1;t<=M;t++)
	{
		alphaij[t] = Lqij[t]>0?1:-1;
	}
	return alphaij;
}
```

****** ldpcBER.cpp ******

```
#include "global.h"
#include <math.h>
int **makeLdpc(int M,int  N, int, int, int onePerCol);
float progress;
float* ldpcBER(int M,int N,int method,int noCycle,int onePerCol,int iter,int frame)
{

	float *ber1,*ber2;
	float *result_Yval;
	char ccc[20];
	int *vhat2 = new int [N+1];
	int **newH = NULL;
	float *EbN0;
	int **H;
	int *bpskMod;
	int *r1,*c1;
	c1 = new int[N+1];
	r1 = new int [M+1];
	float *tx;
	int **dSource;
	int *uk;
	double jjjj = sqrt(2*pi)*N;
	{
		FILE *fp;
		fp = fopen("ldpcBER_log.txt","w");
		fclose(fp);
	}

	int strategy = 2;
	EbN0 = new float[7];
	EbN0[0] = 0;
	EbN0[1] = 0;
	EbN0[2] = 1;
	EbN0[3] = 2;
	EbN0[4] = 3;
	EbN0[5] = 4;
	EbN0[6] = 5;
```

```
int i,j,k,p;
H = makeLdpc(M, N, 1, 1, onePerCol);

/* {
                FILE *fp;

                fp = fopen("ldpcBER_log.txt","a");
                fwrite("------H-----",1,strlen("------H-----"),fp);
                fwrite("\r\n",1,strlen("\r\n"),fp);
                for(int ii = 1;ii<=M;ii++)
                {
                        for(int jj = 1;jj<=N;jj++)
                        {


                                        sprintf(ccc,"%d ",H[ii][jj]);
                                        fwrite(ccc,1,strlen(ccc),fp);


                        }
                        fwrite("\r\n",1,strlen("\r\n"),fp);
                }
                fwrite("\r\n",1,strlen("\r\n"),fp);
                fclose(fp);
        }

*/

int size = 6;//length(EbN0);

ber1 = new float [size+1];
ber2 = new float [size+1];
result_Yval = new float [2*size+1];


int num1;
int num2;
bpskMod = new int [2*M+1];
tx = new float [2*M+1];

dSource = new int*[M+1];
uk = new int [2*M+1];


float **Lqij = new float *[M+1];
int **alphaij = new int *[M+1];

float *Lci = new float [N+1];


for(k = 0;k<=M;k++)
```

```
{
        Lqij[k] = new float [N+1];
        alphaij[k] = new int [N+1];


}
float **Lrji = new float *[M+1];
float **Pibetaij = new float *[M+1];


for( int k = 0;k<=M;k++)
{
        Lrji[k] = new float [N+1];
        Pibetaij[k] = new float [N+1];


}

for(i = 0;i<=M;i++)
{
        dSource[i] = new int [frame+1];
}


int **dSource_param;
dSource_param =new int *[frame+1];
for(k = 0;k<=frame;k++)
{
        dSource_param[k] = new int [M+1];
}
float rat1,rat2;
float N0 ;float xxx;
float yy;int ii;


float **betaij;
betaij = new float *[M+1];
for(int tt = 0;tt<=M;tt++)
{
        betaij[tt] = new float [N+1];
}


int *cr = new int [M+1];
int *tmp1 = new int [M+1];
int *tmp2 = new int [M+1];
int *z= new int [M+1];
int *r = new int [M*N+1];

int *c = new int [M*N+1];

int *r2 = new int [M+1];
```

```
int *c2 = new int [M+1];


        int *vhat1 = new int [N+1];

        int **F = new int*[M+1];

        for(i = 0;i<=M;i++)
        {
                F[i] = new int[N+1];
        }
        int **U = new int *[M+1];
        int **L = new int *[M+1];

        for( int k = 0;k<=M;k++)
        {
                U[k] = new int [N-M+1];
                L[k] = new int [N-M+1];


        }

        int **GG= new int *[M];



        for(i = 0;i<M;i++ )
                {
                        GG[i] = new int [M];
                }

        float **inv_U = new float *[M];
        for( i = 0;i<M;i++)
        {
                inv_U[i] = new float [M];
        }

        int **A = new int *[M+1];
        for(i = 0;i<=M;i++)
        {
                A[i] = new int [M+1];
        }
        int *mul = new int [M+1];

        for (i = 1;i<=size;i++)

        {

                ber1[i] = 0;
                ber2[i] = 0;
```

```
        for(p = 1;p<=M;p++)
        {
                for(int s = 1;s<=frame;s++)
                {
                        int ftemp  = (rand()%RAND_MAX);
                        dSource[p][s] =((float)ftemp /RAND_MAX)>0.5?1:0;
                }

        }

        float cnt;
        if (iter>5)
                cnt=pow(iter1,i/frame1);
        else
                cnt=pow(iter1,(i-1)/frame2);

  for (j = 1;j<=frame;j++)
  {


         for(k = 1;k<=M;k++)
         {
                 dSource_param[j][k] = dSource[k][j];
         }

         //
         cr = makeParityChk(dSource_param[j], H,M,N,
                 strategy,newH,r2,c2,r,c,F,U,L,cr,tmp1,tmp2,z,GG,inv_U,A,mul);
/*       {
                FILE *fp;

                fp = fopen("ldpcBER_log.txt","a");
                fwrite("------c-----",1,strlen("------c-----"),fp);
                fwrite("\r\n",1,strlen("\r\n"),fp);
                for(int ii = 1;ii<=M;ii++)
                {


                                sprintf(ccc,"%d ",cr[ii]);
                                fwrite(ccc,1,strlen(ccc),fp);


                }
                fwrite("\r\n",1,strlen("\r\n"),fp);
                fclose(fp);
         }
*/

         newH = H;
         for(k = 1;k<=M;k++)
         {
                 uk[k] = cr[k];
```

```
            }

            for(k = M+1;k<=(2*M);k++)
            {
                    uk[k] = dSource_param[j][k-M];
            }

/*          {
                    FILE *fp;

                    fp = fopen("ldpcBER_log.txt","a");
                    fwrite("------u-----",1,strlen("------u-----"),fp);
                    fwrite("\r\n",1,strlen("\r\n"),fp);
                    for(int ii = 1;ii<=2*M;ii++)
                    {


                                    sprintf(ccc,"%d ",uk[ii]);
                                    fwrite(ccc,1,strlen(ccc),fp);


                    }
                    fwrite("\r\n",1,strlen("\r\n"),fp);
                    fclose(fp);
            }
*/
            for(k = 1;k<=(2*M);k++)
            {
                    bpskMod[k] = 2*uk[k]-1;
            }
            N0 = (1/(float)(exp(EbN0[i]*log(10.0)/10)));
            for(k = 1;k<=(2*M);k++)
            {
                    xxx = sqrt(N0);///jjjj;
                    xxx *= normalDistribution(rand()%N,N);

                    tx[k] = bpskMod[k] + xxx;
            }
           /* {
                    FILE *fp;

                    fp = fopen("ldpcBER_log.txt","a");
                    fwrite("------tx-----",1,strlen("------tx-----"),fp);
                    fwrite("\r\n",1,strlen("\r\n"),fp);
                    for( ii = 1;ii<=N;ii++)
                    {


                                    sprintf(ccc,"%f ",tx[ii]);
                                    fwrite(ccc,1,strlen(ccc),fp);
```

```
                }
                fwrite("\r\n",1,strlen("\r\n"),fp);
                fclose(fp);


        }
    */
        vhat1 = decodeLogDomain(tx,  newH,  N0,  iter, Lrji, Pibetaij, Lqij, r1, c1, r, c ,Lci,
alphaij, betaij, vhat1);
        /*
        {

                FILE *fp;

                fp = fopen("ldpcBER_log.txt","a");

                fwrite("------vhat1-----",1,strlen("------vhat1-----"),fp);
                fwrite("\r\n",1,strlen("\r\n"),fp);
                for( ii = 1;ii<=N;ii++)
                {


                        sprintf(ccc,"%d ",vhat1[ii]);
                        fwrite(ccc,1,strlen(ccc),fp);


                }
                fwrite("\r\n",1,strlen("\r\n"),fp);
                fclose(fp);


        }
    */
        vhat2 = decodeLogDomainSimple(tx, newH,
iter,Lqij,alphaij,betaij,r1,c1,Lrji,Lci,vhat2);

        /*  {
                FILE *fp;

                fp = fopen("ldpcBER_log.txt","a");
                fwrite("------tx-----",1,strlen("------tx-----"),fp);
                fwrite("\r\n",1,strlen("\r\n"),fp);
                for( ii = 1;ii<=N;ii++)
                {


                        sprintf(ccc,"%f ",tx[ii]);
                        fwrite(ccc,1,strlen(ccc),fp);


                }
                fwrite("\r\n",1,strlen("\r\n"),fp);
```

```
                fwrite("------vhat2-----",1,strlen("------vhat2-----"),fp);
                fwrite("\r\n",1,strlen("\r\n"),fp);
                for( ii = 1;ii<=N;ii++)
                {
                        sprintf(ccc,"%d ",vhat2[ii]);
                        fwrite(ccc,1,strlen(ccc),fp);
                }
                fwrite("\r\n",1,strlen("\r\n"),fp);
                fclose(fp);


        }
        */
          num1 = biterr(vhat1, uk,&rat1);
          ber1[i] = (ber1[i] + rat1);

          num2 = biterr(vhat2, uk,&rat2);
          ber2[i] = (ber2[i] + rat2);
          progress = (float)((i-1)*frame+j)/(size*frame)*100;
        }
  ber1[i] = ber1[i]/(frame*cnt);
  ber2[i] = ber2[i]/(frame*cnt);

}
for(i = 1;i<=size;i++)
        result_Yval[i] = ber1[i];
for(i = size+1;i<=(2*size);i++)
        result_Yval[i] = ber2[i-size];

{
        FILE *fp;

        fp = fopen("ldpcBER_log.txt","a");
        fwrite("------ber1-----",1,strlen("------ber1-----"),fp);
        fwrite("\r\n",1,strlen("\r\n"),fp);
        for(int ii = 2;ii<=6;ii++)
        {


                        sprintf(ccc,"%f ",ber1[ii]);
                        fwrite(ccc,1,strlen(ccc),fp);


        }

        fwrite("\r\n",1,strlen("\r\n"),fp);
        fwrite("------ber2-----",1,strlen("------ber2-----"),fp);
        fwrite("\r\n",1,strlen("\r\n"),fp);
        for( ii = 2;ii<=6;ii++)
        {
                sprintf(ccc,"%f ",ber2[ii]);
```

```
                fwrite(ccc,1,strlen(ccc),fp);
            }
            fwrite("\r\n",1,strlen("\r\n"),fp);
            fclose(fp);

        }

        delete ber1,ber2,result_Yval,vhat1,vhat2,newH,EbN0,H,bpskMod,tx,dSource,uk;


        return result_Yval;
}
double normalDistribution(double x,int N)
{
  int i;
  double a=0,b=x;
  double s,h,sum=0.;
  h=(b-a)/N;
  for( i = 0;i<=N;i++)
  {
        sum = sum + exp(-(a+i*h)*(a+i*h)/2.);

  }
  sum = 0.0+ h*sum/3./sqrt((N-2)*atan(1.));

  return sum;
}
int length(float *EbN0)
{

        int size = _msize(EbN0)/sizeof(EbN0)-1;

        return size;
}
int biterr(int *V,int *F,float *ratio)
{
        int number=0;

        int m = _msize(V)/sizeof(V)-1;
        int max_bits = 0;
        max_bits = V[1];
        int i;
        for( i = 1;i<=m;i++)
        {
            if(V[i] > max_bits)
                max_bits = V[i];
            if(F[i] > max_bits)
                max_bits = F[i];
        }

        max_bits = (log((double)max_bits)/log(2.0))+0.1+1;
        int divide_F = max_bits*m;
```

```
	for( i = 1;i<=m;i++)
	{


		for(int j = 0;j<max_bits;j++)
		{
			if(((V[i]^F[i])>>j)&1)
				number++;
		}
	}
	*ratio = (float)number/divide_F;
	return number;

}
```

******PRMLDlg.cpp******

```
#include "stdafx.h"
#include "PRML.h"
#include "Grafic.h"
#include "PRMLDlg.h"
#include "OptiuniDlg.h"
#include "GenerareDlg.h"
#include "DetaliiPaul.h"
#include "DetaliiCris.h"
#include "DetaliiCatalin.h"
#include "ZgomotDlg.h"
#include "Sxbutton.h"
#include "DetaliiLI.h"
#include "MBTOCDlg2.h"
#include <math.h>
#include "Option.h"

//VARIABILE GLOBALE
	CProgressCtrl m_progresbar;
	BOOL grafic1=FALSE;
	BOOL grafic2=FALSE;
	BOOL grafic3=FALSE;
	int STOP;
	int progres=0;
	CEdit   m_Info;
	CEdit   m_starttimer;
	CEdit   m_endtimer;
	CString         m_SymInfo;
	CSXButton     m_egalizare;
	CSXButton     m_rulare;
	CSXButton     m_optiuni;
```

```
        CSXButton     m_vizual;
        CSXButton     m_eyedg;
        CSXButton     m_STOPSim;
        CSXButton     m_display;
        CSXButton m_generare;
        CSXButton m_exit;
        CSXButton m_despre;
        CSXButton m_bZgomot;
        CSXButton m_bINFO;
        CCanal pCanal(0);

//VARIABILE EXTERNE
extern  int codRLL;
extern  int STOP_FLAG;
extern int m_contorOy;
extern int exceptie;
extern BOOL m_sequence;
extern BOOL m_decoder;
extern int pondere;
extern int real_length;
extern float BER_mediu;
extern int contor_test;
extern BOOL m_test;
extern int file_length;
extern BOOL zgomot;
extern CString detector;
extern CString canal;
extern int m_iter;
extern int m_tipCanal;
extern int m_tipDetector;
extern int m_tipSova;
extern int m_nrPasi;
extern double m_densitatea;
extern double m_RSZ;
extern double m_rata;
extern BOOL m_bEgalizare;
extern BOOL m_bSecventaRLL;
extern int m_RLLD;
extern int m_RLLK;
extern int m_lungFisTemp;
extern int m_lungFisier;
extern int setchannel;
extern double  m_RSZhigh;
extern double  m_RSZlow;
extern double  m_DENShigh;
extern double  m_DENSlow;
extern double m_dispersia;
extern BOOL bIsLDPC;

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
#endif

// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	CStaticm_icon;
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnDetaliiPaul();
	afx_msg void OnDetaliiCatalin();
	afx_msg void OnButton6();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
public:
	afx_msg void OnBnClickedButton7();
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	DDX_Control(pDX, IDC_STATIC_ICON, m_icon);
	//}}AFX_DATA_MAP
}

BOOL CAboutDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	UpdateData();
```

```
	HICON hIcon = AfxGetApp()->LoadStandardIcon(IDI_ASTERISK);
	if(hIcon!=NULL)
		m_icon.SetIcon(hIcon);

	return TRUE;  // return TRUE unless you set the focus to a control
		// EXCEPTION: OCX Property Pages should return FALSE
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
	ON_BN_CLICKED(IDC_BUTTON1, OnDetaliiPaul)
	ON_BN_CLICKED(IDC_BUTTON2, OnDetaliiCatalin)
	ON_BN_CLICKED(IDC_BUTTON6, OnButton6)
	ON_BN_CLICKED(IDC_BUTTON7, OnBnClickedButton7)
	//}}AFX_MSG_MAP

END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CPRMLDlg dialog

CPRMLDlg::CPRMLDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CPRMLDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CPRMLDlg)
	m_tipCanal = m_canal.GetTipCanal();
	m_tipDetector = m_canal.GetTipDetector();
	m_SymInfo = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CPRMLDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CPRMLDlg)
	DDX_Control(pDX, IDC_RADIO_CLASIC, m_bclasic);
	DDX_Control(pDX, IDC_RADIO_CUPRAG, m_bprag);
	DDX_Control(pDX, IDC_RADIO_PR4, m_bpr4);
	DDX_Control(pDX, IDC_RADIO_EPR4, m_bepr4);
	DDX_Control(pDX, IDC_RADIO_E2PR4, m_be2pr4);
	DDX_Control(pDX, IDC_RADIO_DFE, m_bdfe);
	DDX_Control(pDX, IDC_RADIO_VITERBI, m_bviterbi);
	DDX_Control(pDX, IDC_RADIO2, m_bfdts);
	DDX_Control(pDX, IDC_RADIO3, m_bturbo);
	DDX_Control(pDX, IDC_RADIO5, m_bsova);
	DDX_Radio(pDX, IDC_RADIO_CLASIC, m_tipCanal);
	DDX_Radio(pDX, IDC_RADIO_CUPRAG, m_tipDetector);
	DDX_Control(pDX, IDC_PROGRESS2, m_progresbar);
	DDX_Control(pDX, IDC_EDIT2, m_starttimer);
```

```
	DDX_Control(pDX, IDC_EDIT3, m_endtimer);
	DDX_Control(pDX, IDC_EDIT1, m_Info);
	DDX_Text(pDX, IDC_EDIT1, m_SymInfo);
	DDX_Control(pDX, IDC_BUTTON10, m_optiuniLDPC);
	DDX_Control(pDX, IDC_BUTTON9, m_simulareLDPC);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CPRMLDlg, CDialog)
	//{{AFX_MSG_MAP(CPRMLDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(ID_OPTIUNI, OnOptiuni)
	ON_BN_CLICKED(ID_RULARE, OnRulare)
	ON_BN_CLICKED(ID_GENERARE, OnGenerare)
	ON_BN_CLICKED(ID_DESPRE, OnDespre)
	ON_BN_CLICKED(ID_EGALIZARE, OnEgalizare)
	ON_BN_CLICKED(IDC_BUTTON1, OnWaveForm)
	ON_BN_CLICKED(IDC_RADIO_PR4, OnRadioPr4)
	ON_BN_CLICKED(IDC_RADIO_EPR4, OnRadioEpr4)
	ON_BN_CLICKED(IDC_RADIO_E2PR4, OnRadioE2pr4)
	ON_BN_CLICKED(IDC_RADIO_DFE, OnRadioDfe)
	ON_BN_CLICKED(IDC_RADIO_CLASIC, OnRadioClasic)
	ON_BN_CLICKED(IDC_BUTTON2, OnSTOPSim)
	ON_BN_CLICKED(IDC_BUTTON3, OnGrafic)
	ON_BN_CLICKED(IDC_BUTTON4, OnZgomot)
	ON_BN_CLICKED(IDC_BUTTON5, OnButton5)
	ON_BN_CLICKED(IDC_RADIO3, OnTurbo)
	ON_BN_CLICKED(IDC_RADIO_VITERBI, OnRadioViterbi)
	ON_BN_CLICKED(IDC_RADIO2, OnRadio2)
	ON_BN_CLICKED(IDC_RADIO_CUPRAG, OnRadioCuprag)
	ON_BN_CLICKED(IDC_BUTTON7, OnEyeDG)
	ON_BN_CLICKED(IDC_RADIO4, OnRadio4)
	ON_BN_CLICKED(IDC_BUTTON9, OnButton9)
	ON_BN_CLICKED(IDC_RADIO5, OnBnClickedRadio5)
	ON_BN_CLICKED(IDC_BUTTON8, OnBnClickedButton8)
	ON_BN_CLICKED(IDC_BUTTON10, OnButton10)
	//}}AFX_MSG_MAP

END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPRMLDlg message handlers

BOOL CPRMLDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	// Add "About..." menu item to system menu.

//Initializare butoane tip CSXButton (icon&text)
```

```
	m_generare.SubclassDlgItem( ID_GENERARE, this /*parent*/ );
	m_generare.SetIcon( IDI_ICON3, 32, 32 );

	m_egalizare.SubclassDlgItem( ID_EGALIZARE, this /*parent*/ );
	m_egalizare.SetIcon( IDI_ICON3, 32, 32 );

	m_rulare.SubclassDlgItem( ID_RULARE, this /*parent*/ );
	m_rulare.SetIcon( IDI_ICON3, 32, 32 );

	m_STOPSim.SubclassDlgItem( IDC_BUTTON2, this /*parent*/ );
	m_STOPSim.SetIcon( IDI_ICON15, 32, 32 );

	m_exit.SubclassDlgItem( IDCANCEL, this /*parent*/ );
	m_exit.SetIcon( IDI_ICON2, 22, 22 );

	m_optiuni.SubclassDlgItem( ID_OPTIUNI, this /*parent*/ );
	m_optiuni.SetIcon( IDI_ICON1, 32, 32 );

	m_vizual.SubclassDlgItem( IDC_BUTTON1, this /*parent*/ );
	m_vizual.SetIcon( IDI_ICON9, 30, 30 );

	m_eyedg.SubclassDlgItem( IDC_BUTTON7, this /*parent*/ );
	m_eyedg.SetIcon( IDI_ICON17, 30, 30 );

	m_display.SubclassDlgItem( IDC_BUTTON3, this /*parent*/ );
	m_display.SetIcon( IDI_ICON6, 30, 30 );

	m_despre.SubclassDlgItem( ID_DESPRE, this /*parent*/ );
	m_despre.SetIcon( IDI_ICON12, 30, 30 );

	m_bZgomot.SubclassDlgItem( IDC_BUTTON4, this /*parent*/ );
	m_bZgomot.SetIcon( IDI_ICON10, 32, 32 );

	m_bINFO.SubclassDlgItem( IDC_BUTTON5, this /*parent*/ );
	m_bINFO.SetIcon( IDI_ICON16, 32, 32 );

//Stop initializare

	m_vizual.EnableWindow(FALSE);
	m_eyedg.EnableWindow(FALSE);
	m_STOPSim.EnableWindow(FALSE);
	m_rulare.EnableWindow(FALSE);
	m_egalizare.EnableWindow(FALSE);
	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);
	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
```

```
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}
	m_progresbar.SetRange(0,100);
	m_display.EnableWindow(FALSE);
	m_bturbo.SetCheck(1);
	m_optiuniLDPC.EnableWindow(false);
	m_simulareLDPC.EnableWindow(false);

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CPRMLDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CPRMLDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this);	// device context for painting
		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;
```

```
                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
/*
                if(grafic1)
                {
                        CGrafic grafic;
                        grafic.OnPaint();
                }
*/
                CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CPRMLDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CPRMLDlg::OnOptiuni()
{
        m_progresbar.SetPos(0);
        m_starttimer.SetWindowText("                                    ");
        m_endtimer.SetWindowText("                                    ");
        m_SymInfo.Format("                                        ");
        m_Info.SetWindowText(m_SymInfo);
        COptiuniDlg dlgOptiuni(&m_canal);
        if(dlgOptiuni.DoModal()==IDOK)
        {
        }
}

void CPRMLDlg::OnEgalizare()
{
        CString str;
        m_starttimer.SetWindowText("                                    ");
        m_endtimer.SetWindowText("                                    ");
//      UpdateData();

        m_SymInfo.Format("Egalizarea datelor este in curs de desfasurare...");
        m_Info.SetWindowText(m_SymInfo);

        m_canal.SetTipDetector(m_tipDetector);
        m_canal.SetTipCanal(m_tipCanal);
        m_canal.OnEgalizare();
        if(m_tipCanal != CANAL_TURBO) {
                if((int)(ShellExecute(m_hWnd, "open", "textpad.exe",
m_canal.GetNumeFisierEgalizare(), NULL, SW_SHOW))<=32)
```

```
                        AfxMessageBox("Eroare la executia aplicatiei textpad!");
        }
        else {
                if((int)(ShellExecute(m_hWnd, "open", "textpad.exe", "zgomot.txt", NULL,
SW_SHOW))<=32)
                        AfxMessageBox("Eroare la executia aplicatiei textpad!");
        }
        m_rulare.EnableWindow(TRUE);
        m_egalizare.EnableWindow(FALSE);
}

void CPRMLDlg::OnGenerare()
{
        CString str;
        m_starttimer.SetWindowText("                                    ");
        m_endtimer.SetWindowText("                                    ");
        m_SymInfo.Format("Generarea datelor este in curs de desfasurare...");
        m_Info.SetWindowText(m_SymInfo);

//      UpdateData();
        m_canal.SetTipDetector(m_tipDetector);
        m_canal.SetTipCanal(m_tipCanal);

        CGenerareDlg dlgGenerare(&m_canal);
        if(dlgGenerare.DoModal()==IDOK)
        {
                m_canal.Generare();
                m_rulare.EnableWindow(FALSE);
                m_egalizare.EnableWindow(TRUE);
        }
}

void CPRMLDlg::OnDespre()
{
        m_progresbar.SetPos(0);
        m_starttimer.SetWindowText("                                    ");
        m_endtimer.SetWindowText("                                    ");
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
}

void CPRMLDlg::OnRulare()
{
        m_progresbar.SetPos(0);
        CString str;
        if(m_canal.GetTest()==FALSE)
        {
                m_SymInfo.Format("Detectia si decodarea datelor sunt in curs de desfasurare...");
                m_Info.SetWindowText(m_SymInfo);
        }
//      UpdateData();
        m_canal.SetTipDetector(m_tipDetector);
```

```
	m_canal.SetTipCanal(m_tipCanal);

	if(m_canal.GetTest()==FALSE)
	{
		m_canal.Rulare();
		m_rulare.EnableWindow(FALSE);
		m_egalizare.EnableWindow(FALSE);
	}
	else if(m_canal.GetTest()==TRUE)
	{
		m_optiuni.EnableWindow(FALSE);
		m_vizual.EnableWindow(FALSE);
		m_eyedg.EnableWindow(FALSE);
		m_despre.EnableWindow(FALSE);
		if(m_canal.GetFullTest()==FALSE)
		{
			AfxBeginThread(FThread,NULL);
		}
		else if(m_canal.GetFullTest()==TRUE)
		{
			AfxBeginThread(FThread2,NULL);
		}
		m_STOPSim.EnableWindow(TRUE);
	}
}

void CPRMLDlg::OnWaveForm()
{
	// TODO: Add your control notification handler code here
	m_progresbar.SetPos(0);
	m_starttimer.SetWindowText("                                    ");
	m_endtimer.SetWindowText("                                    ");
	UpdateData();
	if(m_tipCanal==CANAL_PRIV)
	{
		if((int)(ShellExecute(m_hWnd, "open", "pr4sim.exe",NULL, NULL,
SW_SHOW))<=32)
			AfxMessageBox("Eroare la executie!");
	}

	else if(m_tipCanal==CANAL_EPRIV)
	{
		if((int)(ShellExecute(m_hWnd, "open", "epr4sim.exe",NULL, NULL,
SW_SHOW))<=32)
			AfxMessageBox("Eroare la executie!");
	}
	else if(m_tipCanal==CANAL_E2PRIV)
	{
		if((int)(ShellExecute(m_hWnd, "open", "e2pr4sim.exe",NULL, NULL,
SW_SHOW))<=32)
			AfxMessageBox("Eroare la executie!");
	}
```

```
	else if((m_tipCanal==CANAL_CLASIC)||(m_tipCanal==CANAL_DFE))
	{
		AfxMessageBox("Forma de unda nu este disponibila!");
	}
}

void CPRMLDlg::OnRadioPr4()
{
	// TODO: Add your control notification handler code here
		m_vizual.EnableWindow(TRUE);
		m_eyedg.EnableWindow(TRUE);
		m_bturbo.SetCheck(0);
		setchannel=0;
		m_tipCanal = CANAL_PRIV;
		//---------SOVA
		m_bsova.SetCheck(0);
		m_bsova.EnableWindow(false);
}

void CPRMLDlg::OnRadioEpr4()
{
	// TODO: Add your control notification handler code here
		m_vizual.EnableWindow(TRUE);
		m_eyedg.EnableWindow(TRUE);
		m_bturbo.SetCheck(0);
		setchannel=0;
		m_tipCanal = CANAL_EPRIV;
		//---------SOVA
		m_bsova.SetCheck(0);
		m_bsova.EnableWindow(false);
}

void CPRMLDlg::OnRadioE2pr4()
{
	// TODO: Add your control notification handler code here
		m_vizual.EnableWindow(TRUE);
		m_eyedg.EnableWindow(TRUE);
		m_bturbo.SetCheck(0);
		setchannel=0;
		m_tipCanal = CANAL_E2PRIV;
		//---------SOVA
		m_bsova.SetCheck(0);
		m_bsova.EnableWindow(false);
}

void CPRMLDlg::OnRadioDfe()
{
	// TODO: Add your control notification handler code here
		m_vizual.EnableWindow(FALSE);
		m_eyedg.EnableWindow(FALSE);
		m_bturbo.SetCheck(0);
```

```
		setchannel=1;
		m_tipCanal = CANAL_DFE;
}

void CPRMLDlg::OnRadioClasic()
{
	// TODO: Add your control notification handler code here
		m_vizual.EnableWindow(FALSE);
		m_eyedg.EnableWindow(FALSE);
		m_bturbo.SetCheck(0);
		setchannel=0;
		m_tipCanal = CANAL_CLASIC;
		//---------SOVA
		m_bsova.SetCheck(0);
		m_bsova.EnableWindow(false);
}

void CPRMLDlg::OnSTOPSim()
{
	// TODO: Add your control notification handler code here
	progres=0;
	STOP=1;
	STOP_FLAG=1;
	m_contorOy=0;
	m_STOPSim.EnableWindow(FALSE);
	m_optiuni.EnableWindow(TRUE);
	if((m_tipCanal==CANAL_PRIV)||(m_tipCanal==CANAL_EPRIV)||(m_tipCanal==CANA
L_E2PRIV)) {
		m_vizual.EnableWindow(TRUE);
		m_eyedg.EnableWindow(TRUE);
	}
	m_despre.EnableWindow(TRUE);
}

void CAboutDlg::OnDetaliiPaul()
{
	// TODO: Add your control notification handler code here
	CDetaliiPaul Paul;
	Paul.DoModal();
}

void CAboutDlg::OnDetaliiCatalin()
{
	// TODO: Add your control notification handler code here
	CDetaliiCatalin Catalin;
	Catalin.DoModal();
}

void CPRMLDlg::OnGrafic()
{
	// TODO: Add your control notification handler code here
	CGrafic *grafic;
```

```
	grafic1=TRUE;
	grafic=new CGrafic(this);
	grafic->Create(IDD_DIALOG3);
	grafic->ShowWindow(SW_SHOW);
	m_display.EnableWindow(FALSE);
}

void CPRMLDlg::OnZgomot()
{
	// TODO: Add your control notification handler code here
	CZgomotDlg zgomot;
	zgomot.DoModal();
}

void CPRMLDlg::OnButton5()
{
	// TODO: Add your control notification handler code here
	if((int)(ShellExecute(m_hWnd, "open", "notepad.exe", "INFO.txt", NULL,
SW_SHOW))<=32)
		AfxMessageBox("Eroare la executia aplicatie notepad!");
}
void CPRMLDlg::OnRadio4()
{
	// TODO: Add your control notification handler code here
	m_vizual.EnableWindow(FALSE);
	m_eyedg.EnableWindow(FALSE);
	m_bclasic.SetCheck(0);
	m_bprag.SetCheck(0);
	m_bdfe.SetCheck(0);
	m_bpr4.SetCheck(0);
	m_bepr4.SetCheck(0);
	m_be2pr4.SetCheck(0);
	m_bviterbi.SetCheck(0);
	m_bfdts.SetCheck(0);
	m_bturbo.SetCheck(0);
	setchannel=2;
	m_tipCanal = CANAL_LDPC;
	m_tipDetector = CANAL_LDPC;
	bIsLDPC = TRUE;
	//---------SOVA
	m_bsova.SetCheck(0);
	m_bsova.EnableWindow(false);
	// m_bpr4.

	m_bviterbi.EnableWindow(false);
	m_bprag.EnableWindow(false);
	m_bfdts.EnableWindow(false);
	m_optiuniLDPC.EnableWindow(true);
	m_simulareLDPC.EnableWindow(true);
}
void CPRMLDlg::OnTurbo()
{
```

```
	// TODO: Add your control notification handler code here
	CString str;
	m_vizual.EnableWindow(FALSE);
	m_eyedg.EnableWindow(FALSE);
	m_bclasic.SetCheck(0);
	m_bprag.SetCheck(0);
	m_bdfe.SetCheck(0);
	m_bpr4.SetCheck(0);
	m_bepr4.SetCheck(0);
	m_be2pr4.SetCheck(0);
	m_bviterbi.SetCheck(0);
	m_bfdts.SetCheck(0);
	setchannel=2;
	m_tipCanal = CANAL_TURBO;
	m_tipDetector = CANAL_TURBO;
	m_bsova.EnableWindow(true);
	m_bsova.SetCheck(0);
	m_tipSova = 0;
	m_bprag.EnableWindow(true);
	m_bviterbi.EnableWindow(true);
	m_optiuniLDPC.EnableWindow(false);
	m_simulareLDPC.EnableWindow(false);

}

void CPRMLDlg::OnRadioViterbi()
{
	// TODO: Add your control notification handler code here
	m_bturbo.SetCheck(0);
	m_tipDetector = DETECTOR_VITERBI;
	//---------SOVA
	m_bsova.SetCheck(0);
	m_bsova.EnableWindow(false);
}
```