DETECTAREA ŞI CORECTAREA ERORILOR LA DECODAREA SECVENŢIALĂ A CODURILOR MATROIDE

G. Bodean, D. Bodean
Universitatea Tehnică a Moldovei

INTRODUCERE

În lucrarea [1] au fost analizate numai aspectele teoretice şi tehnice de codare şi decodare secvenţială a codurilor matroide, fără analiza detectării şi corectării erorilor. Această ”lacună” va fi înlăturată în prezenta lucrare.

Pentru analiza de mai departe acceptăm codarea şi decodarea sistematică. Conform tehnicii standard, aplicată codurilor nonbinare, de exemplu Reed-Solomon, detectarea şi corectarea erorilor (DCE) se execută precum urmează: 1) calculul sindromului; 2) compunerea şi rezolvarea ecuaţiei cheie; 3) calculul valorilor erorilor şi 4) corectarea erorilor.

Calculul sindromului **S** se efectuează conform relaţiei matriciale:

$$\mathbf{S}_{2t\times 1} = \mathbf{H}_{2t\times n} \cdot \mathbf{c}_{n\times 1}. \qquad (1)$$

unde **H** este matricea de control, **c** - vectorul recepţionat, t – numărul simbolurilor de control; n – lungimea cuvintelor de cod.

Valoarea erorii e în poziţia α^i este estimată prin relaţia:

$$e(\alpha^i) = \frac{\Lambda(\alpha^{-i})}{\alpha^i \sigma'(\alpha^{-i})}, \qquad (2)$$

unde:

$$\Lambda(x) = \sum_{i=0}^{t} \lambda_i x^i = \sigma(x) S(x) \bmod x^{2t+1} \qquad (3)$$

este polinomul evaluării erorii, iar $\sigma'(x)$ - derivata abstractă, formală a polinomului locator de erori:

$$\sigma(x) \stackrel{\Delta}{=} \prod_{i=1}^{t} (1 + \alpha^{j_i} x) = 1 + \sum_{i=1}^{t} \sigma_i x^i, \qquad (4)$$

unde $\alpha^i \in \mathbf{GF}(2^m)$; i= 0, 1, …, 2^m-1.

Pentru un polinom arbitrar derivata formală asupra câmpului Galois **GF**(2^m), m = 2, 3, ... , este:

$$f'(z) = \sum_{i=0}^{\psi} a_{2i+1} z^{2i} = a_1 + a_3 z^2 + \ldots + a_{2\psi+1} z^{2\psi}, \qquad (5)$$

unde $\psi = \text{int}\,\frac{n-1}{2}$ [2].

Rădăcinile polinomului (4) ”indică” poziţiile erorilor în vectorul recepţionat **c**. Pentru aceasta poziţiile componentelor vectorului **c** sunt marcate precum este prezentat în fig.1.

poziţia	α^i:	α^0	α^1	α^2	...	α^{n-1}
componenta	c_i :	c_0	c_1	c_2	...	c_{n-1}

Figura 1. Marcarea poziţiilor în cuvântul de cod prin elementele câmpului **GF**(2^m).

Căutarea rădăcinilor polinomului locator $\sigma(x)$ se efectuează prin trierea elementelor nonzero ale câmpului - procedura ***Chien Search***, iar expresia (2) este esenţa algoritmului ***Forney***.

Compunerea polinoamelor $\Lambda(x)$ şi $\sigma(x)$ poate fi efectuată prin una din metodele: 1) algoritmul Berlekamp-Massey (BMA); 2) algoritmul Euclidean (EA); 3) soluţionarea directă sau algoritmul Peterson-Gorenstein-Zierler.

Metoda directă este aplicabilă pentru valori mici ale parametrului t, BMA este rezonabil de implementat la nivel soft, iar EA este larg aplicat în implementarea hard.

Conform tradiţiei, anume algoritmul Euclidean va fi utilizat pentru implementarea algoritmului de detectare şi corectare secvenţială a erorilor în codurile matroide de lungimile n= 2^m şi n= 2^m+1. În continuare vor fi expuse particularităţile DCE pentru aceste coduri şi vor fi prezentate rezultatele implementării codecului matroid în sistemul de proiectare Altera-Quartus.

1. CALCULUL SINDROMULUI

Pentru calculul polinomului sindromului $S(x) = 1 + \sum_{i=1}^{2t} S_i x^i$ al codului matroid cu parametrii (m, k, t), unde:

$$t = (n-k) \textbf{ div } 2,\ n = 2^m \text{ sau } n = 2^m+1 \text{ şi } k < n; \qquad (6)$$

unde m=2, 3, ...; se defineşte matricea de control **H** de forma:

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & \alpha^1 & \ldots & \alpha^{n-2} & 1 \\ 0 & 1 & \alpha^2 & \ldots & \alpha^{2(n-2)} & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 1 & \alpha^{2t-1} & \ldots & \alpha^{(2t-1)(n-2)} & 0 \\ 1 & 1 & \alpha^{2t} & \ldots & \alpha^{2t(n-2)} & 0 \end{bmatrix}, \qquad (7)$$

unde $\alpha^i \in \mathbf{GF}(2^m)$.

Pentru analiza de mai departe ne restricţionăm la cazul $n= 2^m$. Atunci, din matricea (7) se va omite ultima coloană, iar ecuaţia (1) devine egală cu:

$$\begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t-1} \\ S_{2t} \end{bmatrix} = \begin{bmatrix} 0 & 1 & \alpha^1 & \dots & \alpha^{n-2} \\ 0 & 1 & \alpha^2 & \dots & \alpha^{2(n-2)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 1 & \alpha^{2t-1} & \dots & \alpha^{(2t-1)(n-2)} \\ 1 & 1 & \alpha^{2t} & \dots & \alpha^{2t(n-2)} \end{bmatrix} \cdot \begin{bmatrix} c_{-1} \\ c_0 \\ \vdots \\ c_{n-3} \\ c_{n-2} \end{bmatrix}. \quad (8)$$

Din relaţia (8) rezultă:

$$S_{2t} = 1 + \sum_{j=0}^{n-2} c_j \alpha^{2t \cdot j}$$

şi

$$S_i = \sum_{j=0}^{n-2} c_j \left(\alpha^i\right)^j \text{, dacă } i = \overline{1, 2t-1}. \quad (9)$$

Reamintim aici că toate operaţiile aditive şi multiplicative se execută **modulo** polinomul primitiv $p(x)$ al câmpului **GF**(2^m).

Din (9) uşor se observă că la implementare elementul de bază va reprezenta un circuit care trebuie să conţină un multiplicator universal asupra **GF**(2^m) şi un element de memorie. Diagrama acestui circuit de bază este prezentată în fig.2.

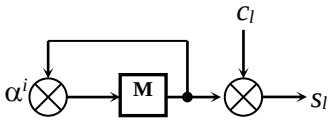

Figura 2. Diagrama circuitului de bază de tip multiplicator-multiplicator: □ *- element de memorie;* ⊗ *- multiplicator.*

Deci, pentru calculul componentei S_i a sindromului $S(x)$ sunt necesare 2^m-2 circuite de bază ajustate la poziţiile unui registru de tip **FIFO** (**F**irst **I**nput, **F**irst **O**utput), în care este stocat vectorul recepţionat **c**. Ieşirile circuitelor de bază, precum rezultă din (9), vor fi conectate bit-cu-bit la un sumator de tip **XOR**.

În fig.3 este prezentată diagrama unităţii de calcul al sindromului pentru cazul $n= 2^m$. Destinaţia multiplexorului **MUX** este uşor de înţeles din relaţia (8); el va fi comutat la poziţia c_{-1} în momentul de timp (tactul) $2t$, cînd se va calcula componenta S_{2t}.

Modul de funcţionare a unităţii. Vectorul recepţionat $c(x)$:

$$c(x) = \sum_{i=0}^{n-1} c_{i-1} x^i, \quad (10)$$

este încărcat secvenţial în registrul **FIFO**. Elementele de memorie ale circuitelor de bază sunt setate în ′1′, selectorul ***sel*** a comutat multiplexorul **MUX** la poziţia ′0′. În perioada de $2t$-1 tacte secvenţial sunt calculate componentele sindromului: S_1, S_2, ... , S_{2t-1}. În ultimul tact, al $2t$-lea, selectorul ***sel*** comutează multiplexorul **MUX** la poziţia c_{-1} şi se formează ultima componentă, S_{2t}, a sindromului $S(x)$.

***Exemplul* 1.** Fie M-codul cu parametrii (n,k,t)= (8, 4, 2) asupra câmpului **GF**(2^3) cu $p(x)= 1+x+x^3$. Vectorul transmis este $c(x)$ = 0. Perturbaţiile în canal au distorsionat două simboluri. În rezultat a fost recepţionat vectorul eronat: $c'(x)=6+x^7 = \alpha^4+\alpha^0 x^7$.

Calculăm sindromul $S(x)$ pentru vectorul eronat, avem:

$$\begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & \boldsymbol{\alpha}^1 & \boldsymbol{\alpha}^2 & \boldsymbol{\alpha}^3 & \boldsymbol{\alpha}^4 & \boldsymbol{\alpha}^5 & \boldsymbol{\alpha}^6 \\ 0 & 1 & \boldsymbol{\alpha}^2 & \boldsymbol{\alpha}^4 & \boldsymbol{\alpha}^6 & \boldsymbol{\alpha}^1 & \boldsymbol{\alpha}^3 & \boldsymbol{\alpha}^5 \\ 0 & 1 & \boldsymbol{\alpha}^3 & \boldsymbol{\alpha}^6 & \boldsymbol{\alpha}^2 & \boldsymbol{\alpha}^5 & \boldsymbol{\alpha}^1 & \boldsymbol{\alpha}^4 \\ 1 & 1 & \boldsymbol{\alpha}^4 & \boldsymbol{\alpha}^1 & \boldsymbol{\alpha}^5 & \boldsymbol{\alpha}^2 & \boldsymbol{\alpha}^6 & \boldsymbol{\alpha}^3 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\alpha}^4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \boldsymbol{\alpha}^6 \\ \boldsymbol{\alpha}^5 \\ \boldsymbol{\alpha}^4 \\ \boldsymbol{\alpha}^6 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 6 \\ 5 \end{bmatrix}. \quad (11)$$

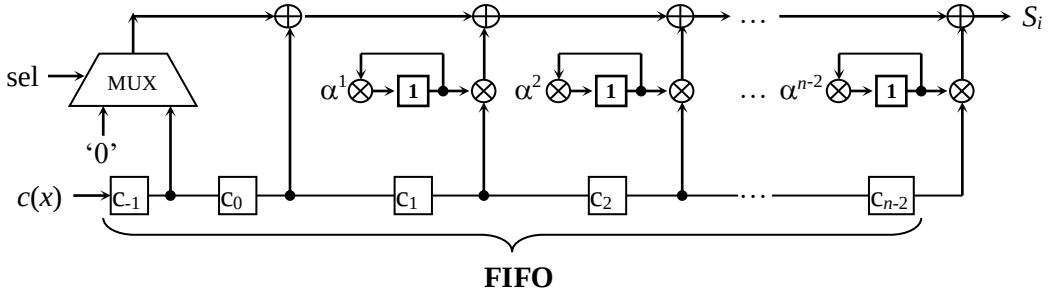

Figura 3. Diagrama unităţii de calcul al componentelor sindromului $S(x)$*:*
□ *- element de memorie setat în ′1′ în momentul iniţial de timp;* ⊕ *- sumator XOR bit-cu-bit.*

Remarca 1: Componentele sindromului din (11) sunt prezentate în forma exponenţială şi numerică.

Deoarece sindromul $S(x)$ nu-i egal cu 0, se trece la compunerea şi rezolvarea ecuaţiei-cheie.

2. ALGORITMUL EUCLIDEAN

În algoritmul Euclidean este calculat cel mai mare divizor comun (CMMDC) a două polinoame $r_0(x)$ şi $r_1(x)$.

Succint, EA conţine următorii paşi:

- Condiţiile iniţiale: $r_0(x)= x^{2t+1}$, $r_1(x)= S'(x)$, $a_0(x)= 1$, $a_1(x)= 0$, $b_0(x)= 0$, $b_1(x)= 1$;
- Pasul $i(i> 2)$, executăm divizarea $r_{i-2}(x)$ la $r_{i-1}(x)$:
- şi calculăm:

$$a_i(x)= a_{i-2}(x)- q_i(x)\, a_{i-1}(x),$$

$$b_i(x)= b_{i-2}(x)- q_i(x) b_{i-2}(x). \qquad (12)$$

- Stop la iteraţia i_{end} în care **deg** $r_{i\text{end}}(x)= 0$.

Atunci CMMDC($r_0(x)$, $r_1(x)$)= $r_u(x)$, unde u este cel mai mare întreg nonzero astfel, încât $r_u(x)\neq 0$ şi $u < i_{\text{end}}$.

Depănăm paşii algoritmului Euclidean pentru exemplul 1. Avem:

- Condiţiile iniţiale:

$r_0(x) = x^5$,
$r_1(x) = S(x) = 1+5x+7x^2+6x^3+5x^4$,
$b_0(x) = 0$,
$b_1(x) = 1$.

- $i= 1$: împărţind $r_0(x)$ la $r_1(x)$, obţinem restul $r_2(x)= 5+5x+7x^2+6x^3$ şi câtul $q_2(x) = 5+2x$. Calculăm:

$$b_2(x)= b_0(x)+ b_1(x)\, q_2(x)= 5+ 2x. \qquad (13)$$

- $i = 2$: împărţim $r_1(x)$ la $r_2(x)$, obţinem:

$r_3(x) = 6x$; $q_3(x)= 2+ 4x$ şi
$b_3(x) = b_1(x)+ b_2(x)\, q_3(x) = 6x+ 3x^2$. (14)

Stop depanarea algoritmului, deoarece **deg** $r_3(x)= 1< t$.Deci, polinomul erorii este $\Lambda(x)= r_3(x)=6x$ şi polinomul locator $\sigma(x)= b_3(x)= 6x+3x^2$.

Unitatea EA conţine două circuite: unul de divizare a polinoamelor şi altul de convoluţie a polinoamelor.

Circuitul de divizare a polinoamelor se va implementa după schema clasică a registrului de deplasare cu sumatoarele incluse, unde legăturile de reacţie vor conţine multiplicatoare la coeficienţii respectivi ai polinomului-numitor $r_i(x)$ [3]. Dar după fiecare pas (iteraţie), valorile coeficienţilor $r_{i-1}(x)$ trebuie să se schimbe (exchange) cu valorile coeficienţilor polinomului-rest r_i (x). Atunci, convenţional diagrama unităţii de divizare a polinoamelor poate fi reprezentată precum este

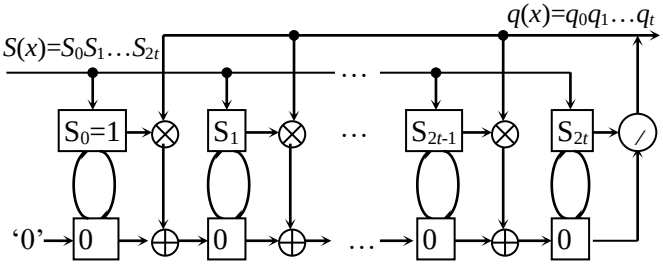

Figura 4. Diagrama circuitului de divizare a polinoamelor: □ - *element de memorie cu starea iniţială indicată;* ⊗ - *multiplicator modular;* ⊘ - *divizor modular.*

arătată în fig.4.

Pentru perfectarea operaţiei de divizare a două polinoame $a(x)$ şi $b(x) <$ **deg** $a(x)\leq 2t+1$ se vor cheltui de la 1 până la $\lceil$ **deg** $a(x)$-**deg** $b(x)\rceil = t$ tacte.

A doua unitate constitutivă a blocului EA este circuitul de convoluţie (înmulţire **modulo** $p(x)$) a două polinoame. Circuitul de convoluţie este destinat pentru calculul polinomului locator $\sigma(x)$ şi poate fi implementat de exemplu, după schema registrului de deplasare cu sumatoare exterioare sau incluse [3]. Însă, fiind vorba de calculul concomitent cu circuitul de divizare, este raţional de implementat circuitul de convoluţie după schema de tip "systolic array" [4].

Convoluţia a două polinoame $a(x) = \sum_{i=0}^{t} a_i x^i$ şi $b(x) = \sum_{i=0}^{t} b_i x^i$ asupra câmpului **GF**(2^m) cu $p(x)$ este un al treilea polinom $c(x)$, astfel încât:

$$c(x) = \sum_{i=0}^{t} c_i x^i = a(x) \cdot b(x) \bmod p(x), \qquad (15)$$

unde $c_i = \sum_{j=0}^{i} a_{i-j} b_j$.

Conform EA în fiecare iteraţie polinomii-factori din (15) se succed unul cu altul, iar la convoluţia $c(x)$ se adaugă rezultatul iteraţiei precedente.

Descrierea neformală efectuată anterior este suficientă pentru a sintetiza circuitul de convoluţie, iar relaţiile (13) şi (14) vor fi ghidul pentru sintetizare diagramei circuitului de convoluţie în exemplul 1, care este prezentată în fig. 5.

Circuitul prezentat în fig.5, funcţionează în modul următor. În momentul iniţial de timp, la începutul depanării EA, registrul **A** conţine coeficienţii polinomului $b_0(x)$, iar registrul **B** -

coeficienţii polinomului $b_1(x)$; starea registrului **D** este arbitrară. Coeficienţii câtului $q(x)$ sunt deplasaţi secvenţial în registrul **Q**.

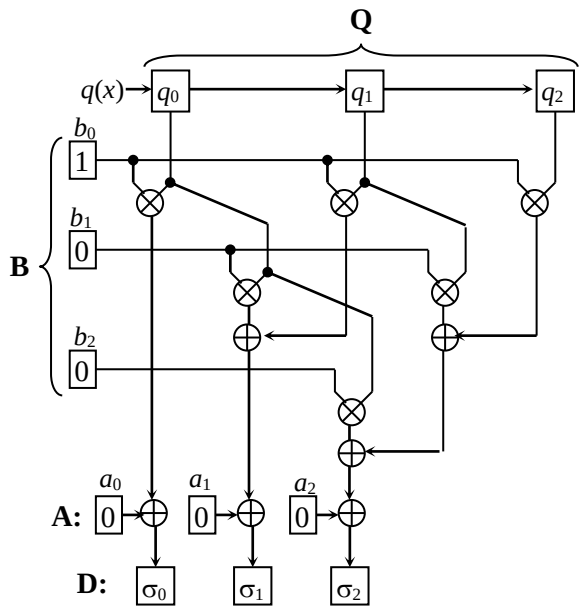

Figura 5. Diagrama circuitului de convoluţie.

Prin tactare, rezultatul convoluţiei şi sumei **B**·**Q**+**A** este salvat în registrul **D**:

$$\mathbf{D} \leftarrow \mathbf{B} \cdot \mathbf{Q} + \mathbf{A}. \qquad (16)$$

Concomitent cu această operaţie (16) poate fi executată operaţie de transcriere a polinomului din **B** în **A**, adică:

$$\mathbf{A} \leftarrow \mathbf{B}. \qquad (17)$$

În final, dacă se execută următorii paşi ai algoritmului Euclidean, atunci are loc transcrierea rezultatului:

$$\mathbf{B} \leftarrow \mathbf{D}. \qquad (18)$$

Deci, în total, în fiecare iteraţie EA (în afară de ultima), circuitul de convoluţie perfectează calculele exact în 2 tacte.

3. LOCALIZAREA ŞI CORECTAREA ERORILOR

Localizarea erorilor se face prin intermediul polinomului locator $\sigma(x)$ care este calculat în EA simultan cu polinomul de evaluare a erorii $\Lambda(x)$. Rădăcinile polinomului $\sigma(x)$ indică poziţiile erorilor în vectorul recepţionat $c(x)$ (vezi fig.1).

Până în prezent nu este cunoscută o metodă de calcul direct al rădăcinilor unui polinom asupra câmpului Galois. De aceea, rădăcinile polinomului sunt căutate prin trierea elementelor nonzero ale câmpului. Procedura este cunoscută sub denumirea ***Chien Search***. Dar procedura *Chien Search* este valabilă pentru cazul $n \le 2^m-1$, adică procedura permite localizarea erorilor componentelor c_i, indicele cărora ia valori din diapazonul $0 \le i \le n-2$, în cazul $n = 2^m$! Rezolvarea problemei e simplă, componenta c_{-1} a vectorului recepţionat rămâne de verificat în afara procedurii, aplicând relaţia $c_{-1} = \sum_{i=0}^{n-2} c_i \alpha^{r \cdot i}$ (vezi (19) din [1]). Şi mai mult, deoarece c_{-1} este un simbol de control, nu este necesară corectarea lui. Dar, din punct de vedere metodic, în continuare vor fi triate toate poziţiile vectorului $c(x)$. Pentru exemplul analizat şi elementele câmpului α^0, α^1, ... , α^6, calculele $\sigma(x)$

Tabelul 1. Calculul valorilor polinoamelor $\sigma(x)$, $\Lambda(x)$ şi $e(x)$.

i	$x = \alpha^{-i}$		$\sigma(\alpha^{-i})=6x+3x^2$	$\lambda(\alpha^{-i})=6x$	$e(\alpha^i)=\frac{\boldsymbol{\alpha}^{-i}\Lambda(\boldsymbol{\alpha}^{-i})}{6}$
	Exp.	Val. num.			
6	$\alpha^{-6}\equiv\alpha^1$	2	7+7=0	7	1
5	$\alpha^{-5}\equiv\alpha^2$	4	5+1=4	5	1
4	$\alpha^{-4}\equiv\alpha^3$	3	1+4=5	1	1
3	$\alpha^{-3}\equiv\alpha^4$	6	2+6=4	2	1
2	$\alpha^{-2}\equiv\alpha^5$	7	4+5=1	4	1
1	$\alpha^{-1}\equiv\alpha^6$	5	3+2=1	3	1
0	α^0	1	6+3=5	6	1
-1	$\alpha^{-\infty}$	0	0+0=0	0	$e(\alpha^{-\infty})$

sunt grupate în Tabelul 1.

Schema circuitului de implementare a polinomului–locator $\sigma(x)$ urmează structura (4) şi va conţine un set de t circuite de bază (fig.2), ieşirile cărora vor fi conectate la un sumator XOR bit-cu-bit. Diagrama circuitului preconizat este prezentată în fig.6.

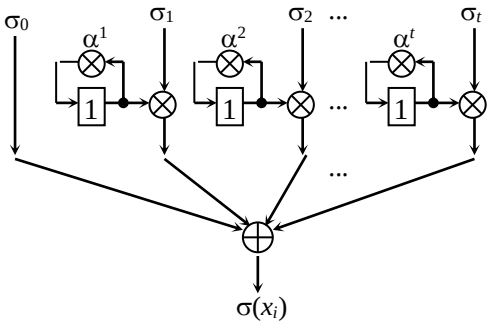

Figura 6. Diagrama circuitului de calcul al polinomului-locator $\sigma(x)$.

În momentul iniţial de timp elementele de memorie ale circuitelor de bază sunt setate în ′1′. Se execută succesiv 2^m-1 tacte ale procedurii *Chien Search*. Sunt fixate poziţiile valorilor i, pentru care a fost determinat:

$$\sigma(\alpha^{-i})= 0. \quad (19)$$

Conform datelor din tabelul 1, rădăcinile polinomului $\sigma(x)$ sunt α^1 şi $\alpha^{-\infty}$, care corespund poziţiilor α^6 şi $\alpha^{-\infty}$, adică componentele c_{-1} şi c_6 sunt eronate.

Valorile erorilor $e(\alpha^i)$ în poziţiile localizate vor fi estimate conform (2). Derivata $\sigma'(x)$, conform (5), este egală cu:

$$\sigma'(x) = (6x+3x^2)' = 6,$$

şi valorile erorilor, conform calculelor din tabelul 1, sunt:

$$e(\alpha^6) = \alpha^{-6}\Lambda(\alpha^{-6})/6 = 7/2\cdot 6 = 1.$$

Eroarea în poziţia $\alpha^{-\infty}$, adică pentru componenta c_{-1}, se va determina conform relaţiei:

$$e\left(\boldsymbol{\alpha}^{-\infty}\right)= \sum_{i=0}^{n-2} c_i \boldsymbol{\alpha}^{2t\cdot i}\ , \quad (20)$$

unde c_i sunt valorile corecte ale componentelor vectorului $c(x)$.

În exemplul analizat, pentru $i = \overline{0,5}$, componentele vectorului recepţionat sunt corecte şi egale cu zero, iar componenta c_6 după corectare, devine:

$$\hat{c}_6 = c_6^{'} + e(\boldsymbol{\alpha}^6) = 1 + 1 = 0.$$

Prin urmare, valoarea componentei $c_{-1} = 0$.

Remarca 2. Reamintim că, de fapt, este *suficient* de căutat rădăcinile pentru indicii componentelor $c(x)$, care reprezintă simbolurile informaţionale. Pentru exemplul analizat avem k=4 simboluri informaţionale localizate pe poziţiile $c_3...c_6$ în vectorul $c(x)$. Aceasta va duce la modificarea respectivă a multiplicatoarelor la constanta α^i din fig.6.

Pentru a definitiva analiza proiectului, trebuie de arătat diagrama circuitului de calcul al polinomului evaluării erorii $\Lambda(x)$. Această diagramă este prezentată în fig.7.

Remarca 3. Proiectul conţine circuite de divizare modulară. Pentru implementarea acestor circuite pot fi aplicaţi algoritmi de generare a unităţii de divizare sau tabelele de transformare a datelor LUT (**L**ook-**U**p-**T**able).

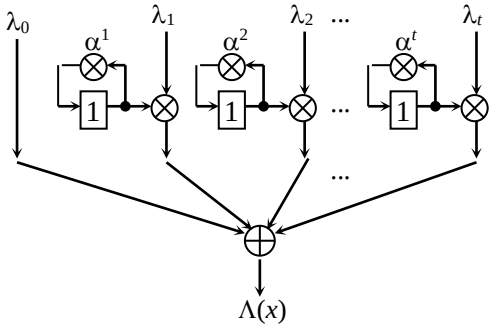

Figura 7. Diagrama circuitului de calcul al polinomului evaluării erorii $\Lambda(x)$.

4. IMPLEMENTAREA UNITĂŢII DE DETECTARE ŞI LOCALIZARE

Platforma soft de proiectare este sistemul Quartus-Altera (version 6). Limbajul de descriere a proiectului este **VHDL**. Schema-bloc integrală a proiectului codecului codului matroid este prezentată în fig.8.

Unităţile de control - *control Coder*, *control Syndrom* şi *control EA*, rămân intacte la adaptarea parametrică a proiectului. De aceea vom analiza resursele *hard* necesare pentru implementarea unităţilor rămase. De regulă complexitatea unui codec cu parametrii (n, k, t) se estimează în raport cu parametrii t şi n, iar în codul matroid avem: $2t=n-k$.

Coderul conţine un registru de deplasare cu reacţie liniară şi un element adiţional. Registrul de deplasare constă din $2t$-1 celule cu binaritatea m biturи. Reacţia liniară conţine $2t$-1 porţi logice

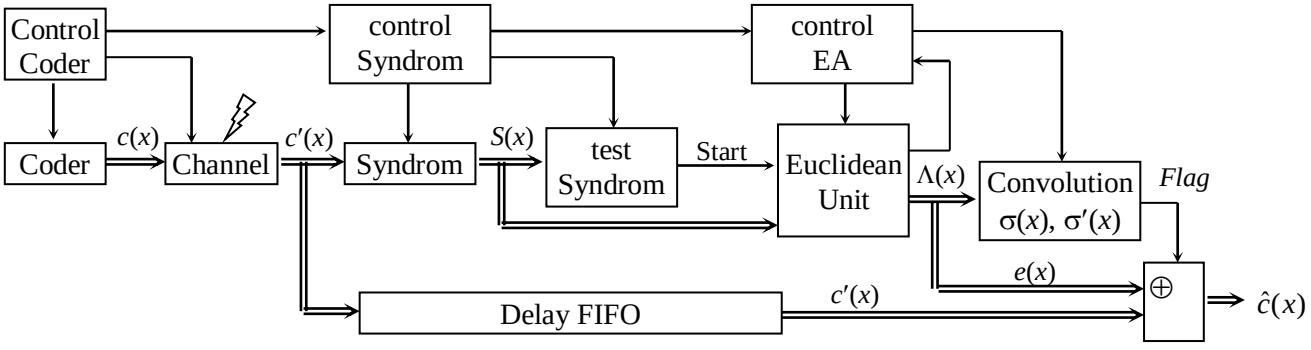

Figura 8. Bloc-diagrama codecului matroid cu unitatea Euclideană de detectare şi corectare a erorilor.

2XOR şi până la 2*t*-1 multiplicatoare la constantă. Elementul adiţional conţine un registru de *m* bituri, un multiplicator universal, un multiplicator la constantă şi *m* porţi **2XOR**.

Blocul *Channel* este destinat pentru simularea zgomotelor (perturbaţiilor) şi nu este o parte inerentă a sistemului.

Unitatea *Syndrom*, conform descrierii anterioare, conţine un registru de deplasare (tip **FIFO**) cu *n* poziţii a câte *m* bituri, *n*-2 elemente de bază complexitatea fiecăruia este egală cu un registru de *m* bituri, un multiplicator la constantă şi un multiplicator universal. Partea combinaţională a unităţii mai conţine *n*- *m* porţi logice **2XOR** şi *m* porţi **2SI**.

Unitatea *TestSyndrom* verifică egalitatea cu zero a componentelor sindromului $S(x)$ şi conţine o poartă logică (*n*-*k*) **SAU** şi (2*t*)· *m* porţi logice **2SI.**

Unitatea EA - unitatea de divizare a polinoamelor şi de calcul a polinomului de evaluare $\Lambda(x)$, respectiv cu complexităţile:

- $2m(n-k)= 4tm$ bituri de memorie, 2*t* multiplicatoare universale şi 2*tm* porţi logice **2XOR**;
- $(t-1)m$ bituri de memorie, *t*-1 multiplicatoare universale şi multiplicatoare la constantă.

Unitatea *Convolution* – unitate de calcul a convoluţiei a două polinoame, a polinomului locator $\sigma(x)$, a derivatei $\sigma'(x)$ şi a valorii erorii $e(x)$. Complexităţile ***hard*** a blocurilor respective sunt:

- calculul convoluţiei – $4(t+1)m$ bituri de memorie, multiplicatoare universale, porţi logice **2XOR;**
- calculul $\sigma(x)$ - *mt* bituri de memorie, *t* multiplicatoare universale şi la constantă, *mt* porţi logice **2XOR;**
- calculul $\sigma'(x)$ - $m((t-1)$ **div** $2)$ bituri de memorie şi porţi logice **2XOR**, $(t-1)$ **div** 2 multiplicatoare universale şi la constantă;
- calculul $e(x)$ - *m* bituri de memorie, *t* multiplicatoare universale şi unul la constantă, un **LUT** de capacitatea 2^m.

În calitate de unitatea *Delay FIFO* poate fi folosit registrul **FIFO** al unităţii *Syndrom*, iar poarta de ieşire **XOR** (fig.8) conţine *m* porţi logice **2XOR**.

Acum pot fi efectuate calculele de estimare prealabilă a complexităţii proiectului pentru a selecta dispozitivul programabil **PLD** potrivit.

În tabelul 2 sunt prezentate formulele de calcul a complexităţilor, iar în tabelul 3 – valorile estimate ale complexităţilor la implementarea *hard* a codecurilor cu parametrii indicaţi.

Tabelul 2. Complexitatea codecului codului matroid.

Memorie, *bit*	$m(12,5t+2n+3/2)+n$
Porţi **2XOR**	$m(t^2/2+6t+n+1/2)$
Porţi **2AND**	$m(2t+1)$
Multiplicatoare la constantă	$n\cdot t/2+n+2$
Multiplicatoare universale	$t^2/2+6t+n+1$

Estimările din ultima linie (marcată cu $m = 8$) a tabelului 3 au o însemnătate practică deosebită, deoarece e vorba de parametrii codecului în standardul DVB-H. Uşor se observă că ponderea cea mai mare în cheltuielile *hard* o au multiplicatoarele universale – de peste 100 de ori mai mare decât alte unităţi ale proiectului. Complexitatea unui multiplicator universal a fost estimată după formula:

$$(2m^2-3m+1)\mathbf{XOR}+(2m^2-m)\mathbf{AND}. \qquad (21)$$

Bineînţeles, instrumentarul de sintetizare a circuitelor combinaţionale CAD-Quartus pot diminua mărimea (21). Dar, necătând la aceasta, complexitatea hard a unui multiplicator universal va rămâne de ordinul $(3...4)m^2$ porţi logice (cu 2 intrări).

În cazul $m= 8$, conform (21) multiplicatorul universal va conţine în total 225 porţi logice şi are o structură ierarhică de cel puţin 15 trepte (etaje). În aceste condiţii reţinerile semnalelor în etajele multiplicatorului vor afecta drastic performanţele de

Tabelul 3. Cheltuielile ***hard*** pentru codecurile cu $k/n = 3/4$.

m	*t*	*n*	*k*	Memorie, *bit*	Multiplicatoare		Porţi **2XOR**	Porţi **2AND**
					la constantă*	universale**		
3	1	8	6	98	30	155+233	45	9
4	2	16	12	250	107	651+864	122	20
5	4	32	24	610	344	2340+2925	325	45
6	8	64	48	1441	1024	7975+9570	867	102
7	16	128	96	3330	2890	27534+32123	2468	231
8	32	256	192	7564	7800	100905+115320	7684	520

*) - complexitatea în medie estimată în porţi logice **2XOR;**

) - complexitatea estimată în porţi **2XOR+2AND.

viteză ale codecului!

Au fost generate proiectele codecurilor pentru caracteristica $k/n=¾$ şi $n=2^m$. Implementarea într-un singur circuit PLD a reuşit pînă la $m=6$. Pentru $m>6$ nici un circuit PLD-Altera nu oferă resurse suficiente pentru implementarea în ansamblu a M-codecului. Soluţia constă în implementarea distribuită a unităţilor prezentate în fig.8.

5. REZULTATELE SIMULĂRII

Pentru testarea experimentală a fost selectat şi implementat M-codecul exemplificat cu parametrii $(n, k, t) = (8, 4, 2)$ asupra $\mathbf{GF}(2^3)$. În fig.9 sunt prezentate diagramele de timp a simulării funcţionării codecului, cazul în care vectorul recepţionat conţine două erori, $c'(x) = x^7+6$.

Explicaţii asupra fig.9. Pentru a atinge o sincronizare cât mai perfectă în circuit se aplică semnale de tactare bifazate. Coderul generează vectorul eronat $c'(x) = x^7+6$ (prima în canal apare componenta c_6). Semnalul *rstSyn* setează elementele de bază ale unităţii *Syndrom* în starea iniţială şi startează calculul sindromului $S(x) = < S_4\ S_3\ S_2\ S_1 >$. deoarece $S(x) \neq 0$, ieşirea *Yes* a unităţii *Syndrom* (fig.8) tranzitează în starea ′1′. Semnalul *Start* porneşte entităţile algoritmului Euclidean:

- La început, instrucţiunea ***MakeDeg***, sunt calculate gradele polinomului–deîmpărţit *DegR* şi a polinomului–divizor *DegD*;
- Instrucţiunea ***Div*** execută un tact de divizare a polinoamelor, la ieşirea *Euclid* apare primul rezultat, egal cu 2, care se înscrie cu semnalul *enConv* în unitatea *Convolution*;
- Instrucţiunea ***Comp*** testează gradele polinoamelor *DegR* şi *DegD*, pentru ca unitatea *controlEuclid* să ia decizia: a continua sau nu divizarea polinoamelor;
- După doi paşi de divizare avem *DegR<DegD*, de aceea are loc schimbul polinoamelor, instrucţiunea ***xch***, şi înscrierea rezultatului convoluţiei câtului $2+5x$ la 1 (instrucţiunea ***Move***);
- Instrucţiunea ***Save*** salvează rezultatul convoluţiei în registrul **B** (fig.5);
- Deoarece *DegR> t*, operaţiile de divizare şi convoluţie a polinoamelor continuă. În rezultat unitatea *Euclidean* (fig.8) va conţine polinomul de

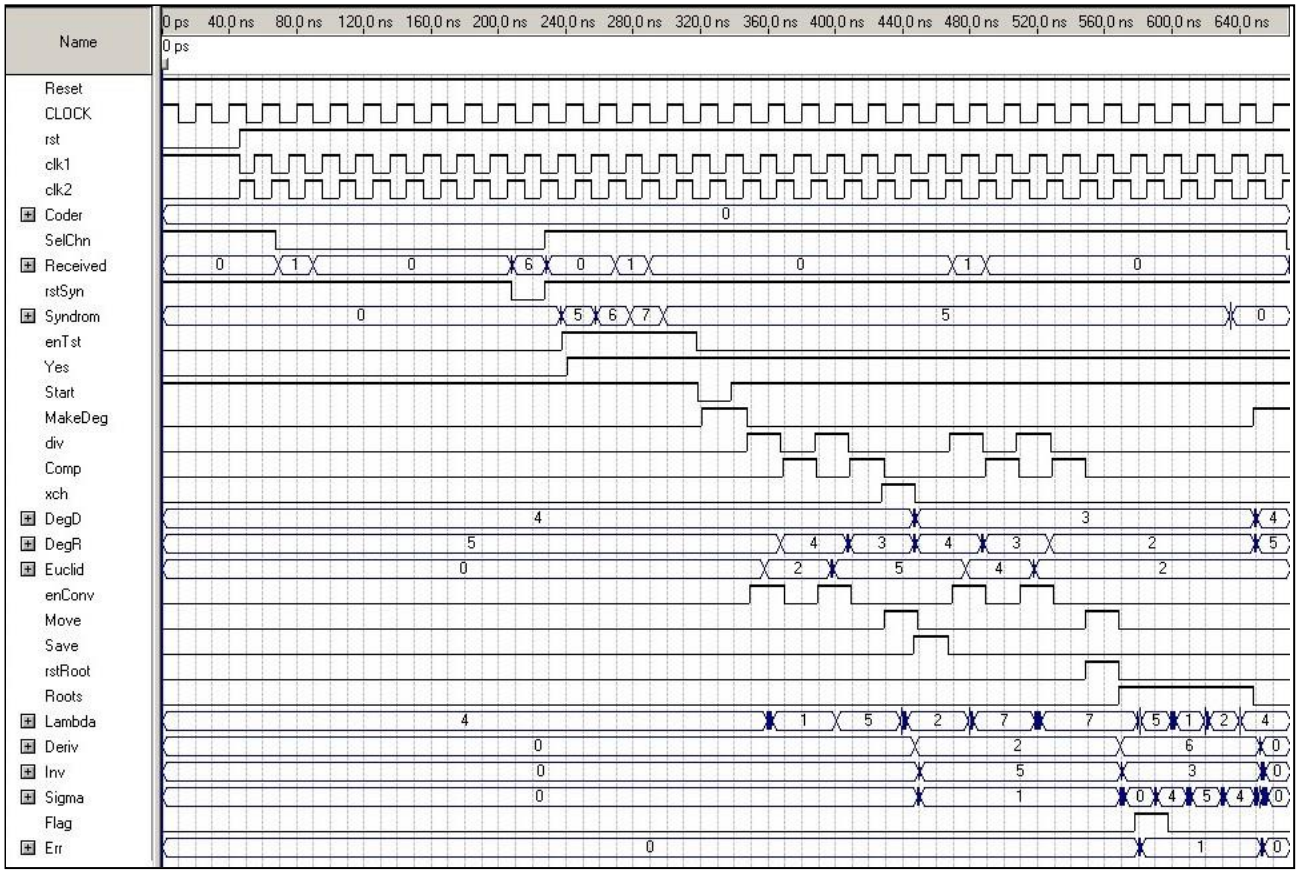

Figura 9. Diagramele waveform a detectării şi localizării erorii duble; codul M(8, 4, 2) asupra $\mathbf{GF}(2^3)$.

Semnalul de permitere *enTst* permite analiza sindromului calculat (vezi linia Syndrom, fig.9), şi, evaluare $\Lambda(x)$, iar unitatea *Convolution*, polinomul-locator $\sigma(x)$ şi derivata acestuia $\sigma'(x)$;

- Instrucţiunea ***rstRoots*** setează în starea iniţială memoriile elementelor de bază, iar în perioada instrucţiunii ***Roots*** are loc calculul valorilor polinoamelor $\Lambda(x)$, $\sigma(x)$, $\sigma'(x)$ şi $e(x)$ pentru primele k= 4 poziţii ale vectorului recepţionat - vezi respectiv liniile *Lambda, Sigma, Deriv* şi *Err* în fig.9.

Precum se observă în fig.9 prin nivelul înalt al semnalului ***Flag*** s-a semnalizat prezenţa erorii în poziţia selectată, iar valoarea erorii ***Err*** pentru această poziţie este egală cu 1, ceea ce corespunde cu rezultatele din tabelul 1.

Trebuie de consemnat faptul că circuitul **PLD**, selectat pentru proiect este din familia **MAX II** de tipul EPM240GT100C3, iar frecvenţa maximală care poate fu atinsă de codec constituie 100 MHz per simbol ori 400 MHz per bit.

CONCLUZII

Elaborarea proiectelor codecurilor codurilor nonbinare corectoare de erori, în deosebi ale codurilor Reed-Solomon, este o temă închisă pe ”arena” telecomunicaţiilor în ultima perioadă de timp. Această bătălie este iniţiată, în mare parte, de acceptarea standardelor DVB, în particular DVB-H, care impune o rată înaltă de prelucrare a datelor şi o performanţă considerabilă a capacităţii corectoare t, egală cu 32. Însă standardul DVB este orientat anume spre aplicarea RS-codurilor, lungimea cărora este limitată $n \leq 2^m-1$. În anumite domenii, şi anume reţelele de calculatoare, sistemele **RAID** etc., această restricţie este inacceptabilă. Bineînţeles că există modificări ale codurilor Reed-Solomon, aşa numitele, RS-coduri *extinse*, care, cel puţin la nivel algoritmic, propun soluţii şi pentru cazurile $n= 2^m$ şi $n= 2^m+1$.

Pe de altă parte, în silinţa lor de ”a fi printre primii”, autorii publicaţiilor lasă în afara analizei unele probleme, care, la prima vedere, ar părea să fie minore, dar care pot avea repercusiuni majore asupra performanţelor proiectului. Revenind asupra algoritmului Euclidean, este vorba de analiza algoritmului de control al procedurii de divizare a polinoamelor. Doar, în practică, pot fi cazuri diferite! De exemplu, **deg** $S(x)< 2t$, sau **deg** $r(x)\leq t$ imediat după executarea primei instrucţiuni de divizare, sau numărul de paşi de divizare variază de la o iteraţie la alta (de la 1 pînă la $2t+1$ paşi) etc. Aceste ”particularităţi” se răsfrâng asupra timpului de divizare a polinoamelor şi pot introduce o reţinere (*Delay*) considerabilă în procesul de detectare şi localizare a erorilor.

Atunci, rămâne ca proiectantul să decidă (aleagă) ce tip de arhitectură este mai convenabilă la moment pentru implementarea codecului: ori *asincronă*, ori *secvenţială*, ori cea *mixtă*, adică secvenţia-asincronă.

Pentru detectarea erorilor (calculul sindromului) în codurile Reed-Solomon se foloseşte matricea de control care urmează structura matricei Vandermonde. Pe de altă parte, matricea Vandermonde şi modificările sale care păstrează proprietatea de independenţă liniară reprezintă matricea matroidului uniform asupra câmpului Galois. De aceea, prin extensie, se pot afirma că RS-codurile sunt o subclasă a codurilor matroide.

În lucrare a fost prezentată tehnica (metodica) de aplicare a algoritmului Euclidean de divizare a polinoamelor pentru localizarea şi corectarea erorilor în codurile matroide cu procesarea secvenţială simbol cu simbol a datelor. Metodica propusă este adusă la nivelul de implementare **VHDL**–proiect. Verificările şi testările efectuate au demonstrat viabilitatea proiectului. Prin adaptare, proiectul elaborat poate genera structuri ale codecurilor pentru o gamă largă şi variată a parametrilor codurilor matroide corectoare de erori.

Bibliografie

1. Bodean G*. Codarea şi decodarea secvenţială a codului matroid.- Meridian Ingineresc, Nr. 2, Chişinău, 2007.*

2. Mutter V.M. *Osnovy pomehonstojchivoj teleperedachi informaczii. –L.: Energoatomizdat. Leningr. otd.-nie., 1990, 228 p.*

3. Peterson W.W., Weldon E.J. *Error-Correcting Codes . - MIT Press, Cambridge, 1972.- 560 p.*

4. Ullman J. *Computational aspects of VLSI. - New York, NY: Computer Science Press, 1984.*

Aprobat spre publicare:20.01.2007