ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΨΗΦΙΑΚΕΣ ΕΠΙΚΟΙΝΩΝΙΕΣ ΙΙ

2η ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ

Ειρήνη Δόντη

ΑΜ: 03119839

9ο εξάμηνο

Αθήνα 2023-2024

1. **Εύρεση κατάλληλου Κυκλικού κώδικα BCH (15,k) με δυνατότητα διόρθωσης 3 λαθών (δηλαδή πρέπει να βρεθεί κατάλληλο k και το γενετήριο πολυώνυμο του κώδικα).**

   a. n = 15 = 16 – 1 = 2^4 – 1 ή m = 4. Γνωρίζουμε ότι t =3 και n – k ≤ mt ή k ≥ 3. Δεδομένου ότι t = 3, το γενετήριο πολυώνυμο που ψάχνουμε θα είναι το πολυώνυμο ελάχιστου βαθμού με ρίζες τα α, α^2, α^3, …, α^6 του σώματος GF(2^4). Σύμφωνα με τη δοθείσα σχέση g(x) = ΕΚΠ{φ_1(x), φ_3(x),..., φ_5(x)}, όπου φ_i(x) το ελάχιστο πολυώνυμο του στοιχείου α^i.
   Ανατρέχοντας στον πίνακα του πλαισίου Π.4. προκύπτει ότι:
   g(x) = $(x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)$ = $(x^8 + x^7 + x^6 + x^5 + x^5 + x^4 + x^4 + x^4 + x^3 + x^3 + x^2 + x^2 + x + x + 1)(x^2 + x + 1)$ = $x^{10} + x^9 + x^9 + x^8 + x^8 + x^8 + x^7 + x^7 + x^7 + x^7 + x^6 + x^6 + x^6 + x^6 + x^6 + x^6 + x^5 + x^5 + x^5 + x^5 + x^5 + x^5 + x^4 + x^4 + x^4 + x^4 + x^4 + x^4 + x^4 + x^3 + x^3 + x^3 + x^3 + x^3 + x^3 + x^2 + x^2 + x^2 + x^2 + x^2 + x + x + x + 1 = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$.
   Το πολυώνυμο που προκύπτει είναι $10^{ου}$ βαθμού και συνεπώς n-k = 10, k = 5.

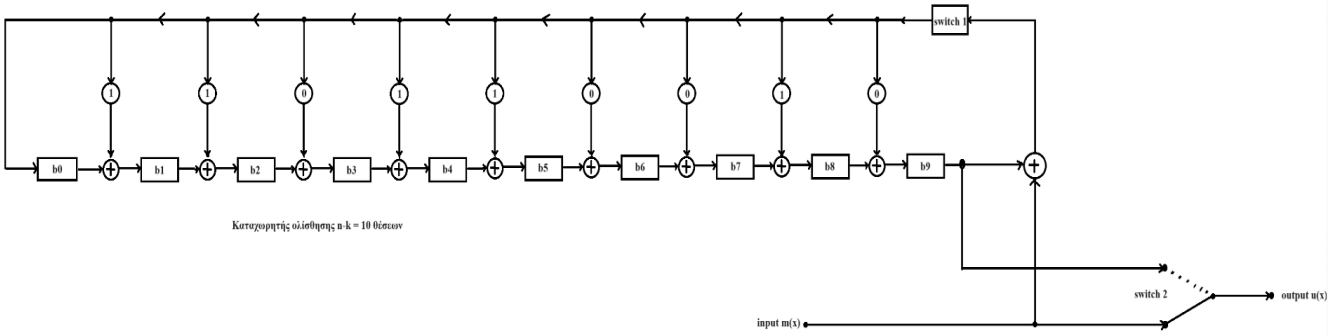
   b. Πράγματι, για t = 3 και n = 15, ο πίνακας δίνει τιμές k = 5 και g(x) = 2467 = $2 \cdot 8^3 + 4 \cdot 8^2 + 6 \cdot 8^1 + 7 \cdot 8^0$ (σε οκταδικό σύστημα). Για την επαλήθευση της ταύτισης των αποτελεσμάτων το μετατρέπουμε στο δυαδικό σύστημα: 2467 = $2 \cdot 8^3 + 4 \cdot 8^2 + 6 \cdot 8^1 + 7 \cdot 8^0 = 2^{10} + 2^8 + 3 \cdot 2^4 + 7 \cdot 2^0$ = 10100110111 → $x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$
   Όμως, το δεξιότερο bit σε αυτήν τη τιμή αντιστοιχεί στο σταθερό όρο, συνεπώς ξεκινώντας από τα δεξιά προς τα αριστερά προκύπτει το ίδιο γενετήριο πολυώνυμο με πριν: g(x) = $x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$.

2. Υλοποίηση Κώδικα BCH.

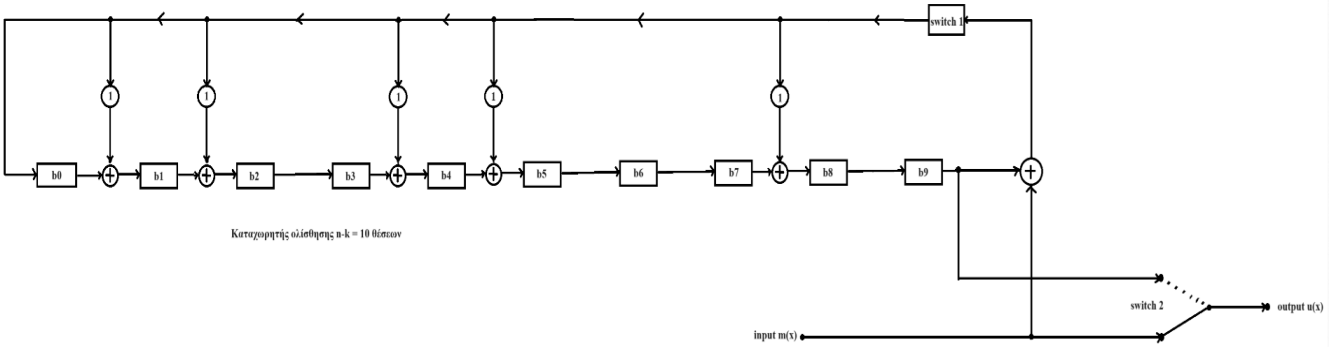
a.

Για τον κώδικα BCH(15, 5), ανατρέχοντας στον πίνακα της σελίδας 9-24, προκύπτει ότι g(x) = $x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$, όπως αναφέρθηκε παραπάνω.

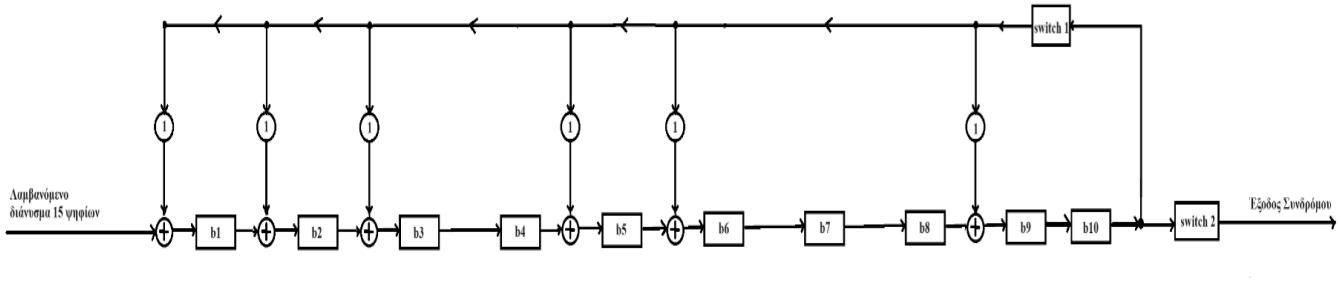
Σχεδιάζουμε το κύκλωμα υλοποίησης του παραπάνω κώδικα BCH(15, 5), ξεκινώντας από τον πρώτο μη σταθερό όρο αριστερά προς τα δεξιά, με βάση το γενικό κύκλωμα των κωδίκων του Πλαισίου 9.15.



Έπειτα από απλοποίηση, το κύκλωμα που προκύπτει είναι:



b. Σχεδιάζουμε το κύκλωμα υπολογισμού του συνδρόμου με βάση το γενικό κύκλωμα του Πλαισίου 9.16.



Υλοποιούμε το συνολικό κύκλωμα του Αποκωδικοποιητή με βάση το γενικό κύκλωμα του Πλαισίου 9-8.

Η έξοδος που θα λάβουμε, αποτελείται από τα 10 bits $S = [s_1, s_2, \ldots, s_9, s_{10}]$ του συνδρόμου.

Αντιστοιχίζουμε τα σύνδρομα με τα αντίστοιχα λάθη με κύκλωμα παρόμοιο με του πλαισίου 9.8(β) μόνο που θα έχουμε 10 παράλληλα καλώδια, αφού έχουμε 10 bits συνδρόμου και 15 XOR πύλες καθώς το μήκος της κωδικολέξης είναι 15 στην προκειμένη περίπτωση.

Από την παραπάνω ανάλυση, προκύπτει πίνακας με 2^{10} = 1024 σύνδρομα. Τα διανύσματα λάθους με διορθώσιμα λάθη είναι 1 (0 bit λάθος)+15 (1 bit λάθος) +$\binom{15}{2}$ (2 bit λάθος) + $\binom{15}{3}$ (3 bit λάθος) = 15 + 105 + 1 + 455 = 576 διανύσματα λάθους με μέγιστο 3 bit διορθώσιμα λάθη. Προσθέτουμε 1024-576 = 448 διανύσματα λάθους με 4 bit λάθος. Επιλέγουμε την κωδικολέξη 000000000000000 και προσθέτουμε τα διανύσματα λάθους με τη σειρά. Οπότε, προκύπτει το πολυώνυμο r(x) ($14^{ου}$ βαθμού). Διαιρούμε το πολυώνυμο r(x) με το γενετήριο πολυώνυμο g(x) και το υπόλοιπο είναι το σύνδρομο ($9^{ου}$ βαθμού) που αντιστοιχεί στο εκάστοτε διάνυσμα λάθους. Για τη σωστή αντιστοιχία, συμβουλευόμαστε τον πίνακα που κατασκευάσαμε παρακάτω και οι πύλες AND που οδηγούν στις πύλες XOR θα έχουν την ακόλουθη αντιστοιχία:

Σύνδρομα	**Διάνυσμα Λαθών e**
0000000000	000000000000000
1010011011	100000000000000
0101001101	110000000000000
1101110000	101000000000000
0011110101	100100000000000
1110101100	100010000000000
0010011011	100001000000000
1110011011	100000100000000
1000011011	100000010000000
1011011011	100000001000000
1010111011	100000000100000
1010001011	100000000010000
1010010011	100000000001000
1010011111	100000000000100
1010011001	100000000000010
1010011010	100000000000001
…	…

3. Εξομοίωση/υλοποίηση της λειτουργίας Κωδικοποιητή και Αποκωδικοποιητή του παραπάνω ερωτήματος στο MATLAB.

Παρακάτω, παραθέτουμε τον κώδικα προσομοίωσης της λειτουργίας Κωδικοποιητή και Αποκωδικοποιητή με τη βοήθεια κώδικα MATLAB:

```
% Eirini Donti 03119839
clear all; close all;
m=4; n=2^m-1; t=3; k=5;
g= gf([1 0 1 0 0 1 1 0 1 1 1],1); % BCH (15,5) - trivial gf
msg1 = gf(zeros(k,1)',1);
msg2 = gf(ones(k,1)',1);
msg3 = gf([1 0 1 0 1],1);         % random msg
%% verify that with the same 3-bit error the syndrome is the same for all
three messages
u1 = [msg1 zeros(n-k,1)']; % codeword
[q,b1] = deconv(u1,g);     % remainder b1
u1 = u1 + b1;
u2 = [msg2 zeros(n-k,1)'];
[q,b2] = deconv(u2,g);
u2 = u2 + b2;
u3 = [msg3 zeros(n-k,1)'];
[q,b3] = deconv(u3,g);
u3 = u3 + b3;
e = gf([ones(t,1)',zeros(n-t,1)'],1); % random error e
r1 = u1 + e;
r2 = u2 + e;
r3 = u3 + e;
[q br1] = deconv(r1,g); % syndrome br1
[q br2] = deconv(r2,g); % syndrome br2
[q br3] = deconv(r3,g); % syndrome br3
%last 10 bits is syndrome
%br1 + br2
%br1 + br3
%% --------------
% Form the syndrome-error look_up table
% 2^10 = 1024 syndromes with the corresponding errors
% n = 15 single-error patterns nx(n-1)/2 = 105 double-error pattern
% ---------------
%% --- Try the single- and double-error patterns
zer = zeros(n,1)';
E=[]; S=[]; j=0;
for i=1:n
    e1=zer;e1(i) = 1;
    E=[E;e1];
    r1 = u1+e1;
    [q br] = deconv(r1,g);
    S=[S;br(k+1:end)];
    j=j+1;
    e2 = e1;
    for kk=1:n
        ee=e2;
        if e2(kk)==0 ee(kk)=1; end
        flag=0; J=j;
        for l=1:J;
            if E(l,:)==ee flag=1; break;end
        end
        if flag==0;
            E=[E;ee];
            r1=u1+ee;
            [q br] = deconv(r1,g);
            S=[S;br(k+1:end)];
```

```
            j=j+1;
        end
    end
end

%%-----Now,with all,three patterns
zer=zeros(n,1)';
E=[];S=[];j=0;
for i=1:n
    e1=zer;e1(i)=1;
    E=[E;e1];
    r1=u1+e1;
    [q br]=deconv(r1,g);
    S=[S;br(k+1:end)];
    j=j+1;
    e2=e1;
    for ii=1:n
        ee=e2;
        if e2(ii)==0 ee(ii)=1; end
        flag=0; J=j;
        for l=1:J
            if E(l,:)==ee flag=1; break; end
        end
        if flag==0
            E=[E;ee];
            r1=u1+ee;
            [q br] = deconv(r1,g);
            S=[S;br(k+1:end)];
            j=j+1;
        end
        e3=ee;
        for iii=1:n
            eee=e3;
            if e3(iii)==0 eee(iii)=1; end
            flag=0; J=j;
            for l=1:J
                if E(l,:)==eee flag=1;break;end
            end
            if flag==0
                E=[E;eee];
                r1=u1+eee;
                [q br]=deconv(r1,g);
                S=[S;br(k+1:end)];
                j=j+1;
            end
        end
    end
end

%%------------ Tests ---------
r3=u3+E(end-3,:);
[q br3]=deconv(r3,g);
br3(k+1:end)+S(end-3,:);

%%------ Add zero error and 448 4-bit error patterns
E=[zeros(n,1)'; E]; S=[zeros(n-k,1)';S]; %0 bit error
J=length(E); JJ=min(n*(n-1)*(n-2)/2/3,2^(n-k)-J);
jj=1;
for j=1:J
    if sum(E(j,:))==3
        eeee=E(j,:);
        for i=1:length(eeee)
            if eeee(i)==0
                eeee(i)=1;
                break;
            end
```

```
            end
            E=[E;eeee];
            r1=u1+eeee;
            [q br]=deconv(r1,g);
            S=[S;br(k+1:end)];
            jj=jj+1;
        end
        if jj>JJ break; end
    end
```