ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αναφορά 2ης Εργαστηριακής Άσκησης στις Ψηφιακές Επικοινωνίες ΙΙ

«Σχεδίαση και Υλοποίηση Δυαδικού Κώδικα BCH»

Μπουφίδης Ιωάννης 03120162

Ερώτημα 1

α) Έστω Κυκλικός Κώδικας BCH μήκους κωδικολέξης $\boldsymbol{n = 15}$ και με δυνατότητα διόρθωσης $\boldsymbol{t = 3}$ λαθών. Από την σχέση $\boldsymbol{n = 2^m - 1}$, προκύπτει ότι $\boldsymbol{m = 4}$. Επομένως, για να βρούμε το γενετήριο πολυώνυμο $g(x)$ του παραπάνω κώδικα, πρέπει να υπολογίσουμε το ελαχίστου βαθμού πολυώνυμο με ρίζες τα στοιχεία $\boldsymbol{a, a^2, a^3, \dots, a^{2t=6}}$ του σώματος GF($2^{m=4}$), με a πρωταρχικό.

Εφαρμόζοντας την σχέση $g(x) = ΕΚΠ\{\varphi_1(x), \varphi_3(x), \dots, \varphi_{2t-1}(x)\}$ και συμβουλευόμενοι τον παρακάτω πίνακα, ο οποίος δίνει τις ομάδες των συζυγών στοιχείων και τα αντίστοιχα ελάχιστα πολυώνυμα του GF(2^4), προκύπτει ότι το γενετήριο πολυώνυμο του παραπάνω κώδικα ισούται με

$$\boldsymbol{g(x) = (X^4 + X + 1)(X^4 + X^3 + X^2 + X + 1)(X^2 + X + 1)}$$

συζυγείς ρίζες	ελάχιστα πολυώνυμα
0	X
1	$X+1$
a, a^2, a^4, a^8	X^4+X+1
a^3, a^6, a^9, a^{12}	$X^4+X^3+X^2+X+1$
a^5, a^{10}	X^2+X+1
$a^7, a^{11}, a^{13}, a^{14}$	X^4+X^3+1

Επίσης, γνωρίζουμε ότι το πλήθος των bits ισοτιμίας $(n - k)$ ισούται με τον βαθμό του γενετηρίου πολυωνύμου. Επομένως, για τον προκείμενο κώδικα, έχουμε ότι τα bits του μηνύματος κάθε κωδικολέξης είναι

$$n - k = 15 - k = 10 \rightarrow \boldsymbol{k = 5}\ \textbf{bits}.$$

Άρα, ο κυκλικός κώδικας BCH της άσκησης είναι ο **(15, 5)**.

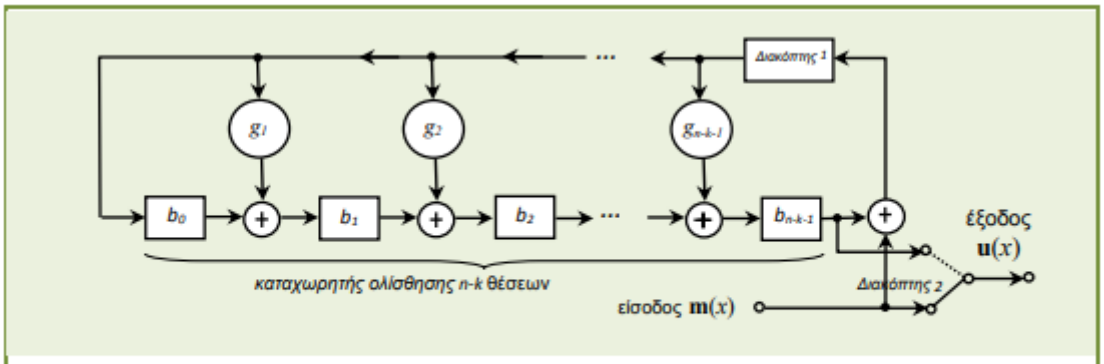
β) Παρατηρώντας τον διπλανό πίνακα, μπορεί κανείς να επιβεβαιώσει ότι, πράγματι, ο Κυκλικός Κώδικας BCH μήκους κωδικολέξης $\boldsymbol{n = 15}$ και με δυνατότητα διόρθωσης $\boldsymbol{t = 3}$ λαθών, έχει μήκος μηνύματος $\boldsymbol{k = 5}$.

Επίσης, παρατηρούμε ότι $g(x) = (2\ 4\ 6\ 7)_{10} = (010\ 100\ 110\ 111)_2$. Δηλαδή, $\boldsymbol{g(x) = X^{10} + X^8 + X^5 + X^4 + X^2 + X + 1}$, αποτέλεσμα που ταυτίζεται με το γενετήριο πολυώνυμο που υπολογίσαμε στο υποερώτημα **(α)**.

n	k	t	$g(x)$
7	4	1	13
15	11	1	23
	7	2	721
	5	3	2467
31	26	1	45
	21	2	3551
	16	3	107657
	11	5	5423325
	6	7	313365047
63	57	1	103
	51	2	12471
	45	3	1701317
	39	4	166623567
	36	5	1033500423
	30	6	157464165547
	24	7	17323260404441
	18	10	1363026512351725
	16	11	6331141367235453
	10	13	47262230552725015 5
	7	15	52310455435032717 37

Ερώτημα 2

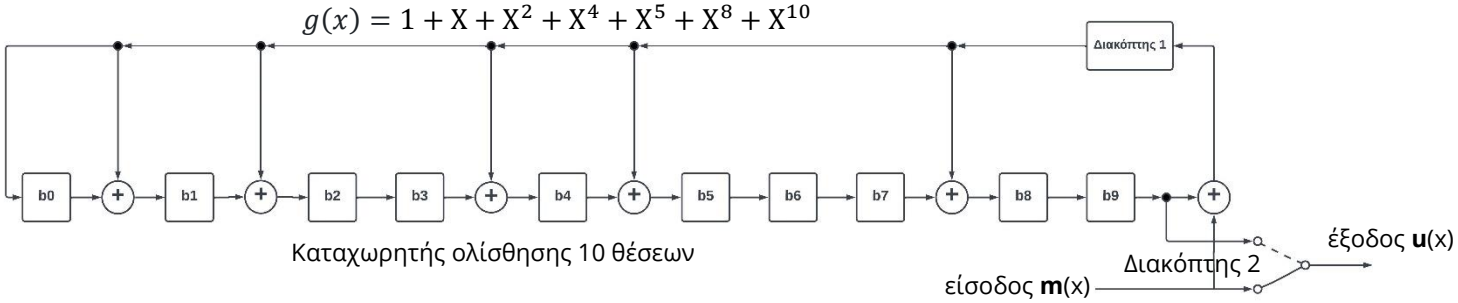
α) Με βάση το γενικό κύκλωμα των κωδίκων του Πλαισίου 9.15, το οποίο φαίνεται παρακάτω,



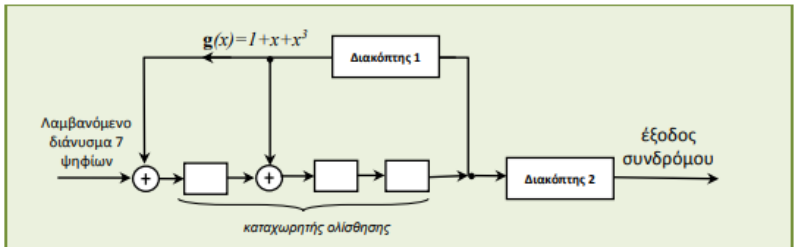
Πλαίσιο 9.15: Κύκλωμα υλοποίησης του (n,k) κυκλικού κώδικα με γενετήριο πολυώνυμο το $\mathbf{g}(x) = 1 + g_1 x + g_2 x^2 + \ldots + g_{n-k-1} x^{n-k-1} + x^{n-k}$

σχεδιάζουμε το κύκλωμα υλοποίησης του κώδικα BCH (15, 5) του **Ερωτήματος 1**, ως εξής:

$$g(x) = 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}$$



β) Στην συνέχεια, χρησιμοποιώντας το σχήμα του Πλαισίου 9.16,

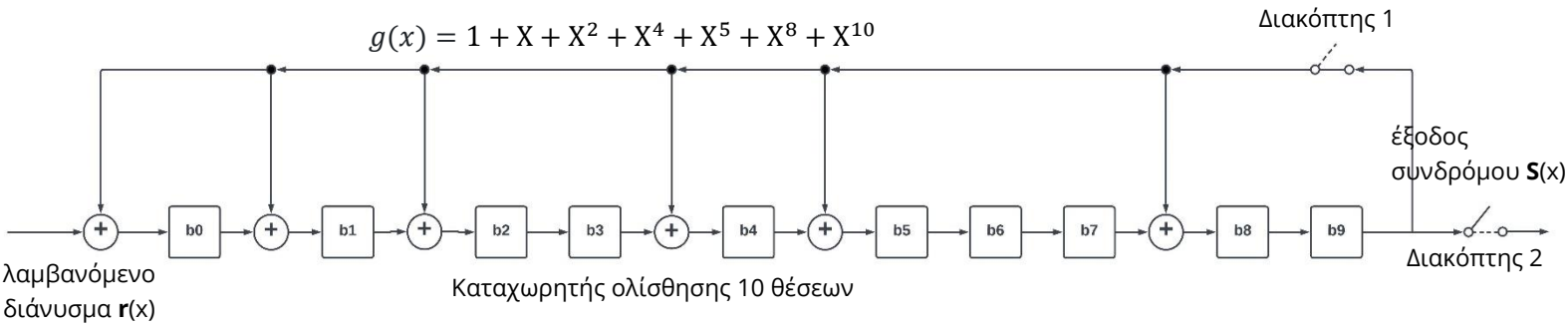


*Ο Διακόπτης 1 είναι αρχικά κλειστός και ο Διακόπτης 2 ανοικτός. Το λαμβανόμενο διάνυσμα εισέρχεται σταδιακά στον καταχωρητή εισόδου, ο οποίος είναι αρχικοποιημένος στο 0. Αφού εισέλθει **ολόκληρο** το λαμβανόμενο διάνυσμα στον καταχωρητή ολίσθησης, τα περιεχόμενα του καταχωρητή είναι το σύνδρομο. Τότε ο Διακόπτης 1 ανοίγει και ο 2 κλείνει, ώστε να εξαχθεί σειριακά το σύνδρομο.*

Πλαίσιο 9.16: Κύκλωμα υπολογισμού συνδρόμου του κυκλικού κώδικα (7,4) του Πλαισίου 9.13

μπορούμε να σχεδιάσουμε το αντίστοιχο κύκλωμα υπολογισμού του συνδρόμου ως εξής:

$$g(x) = 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}$$



Εφόσον, το σύνδρομο υπολογίζεται από τον τύπο $\boldsymbol{s = rH^T}$, όπου $\boldsymbol{H^T}$: η ανάστροφη της μήτρας ελέγχου ισοτιμίας με διάσταση $(n = 15)x((n - k) = 10)$, το σύνδρομο έχει μέγεθος $n - k = 10$ bits.

Μπορούμε να υπολογίσουμε ότι τα διανύσματα λάθους μήκους $n = 15$ bits:

- με 1 bit μονάδα είναι **15 διανύσματα**
- με 2 bits μονάδες είναι $\boldsymbol{14 + 13 + 12 + \ldots + 1 = \sum_{i=1}^{n-1} i =}$ **105 διανύσματα**
- με 3 bits μονάδες είναι $\boldsymbol{(13 + 12 + \cdots + 1) + (12 + 11 + \cdots + 1) + \cdots + 1 = \sum_{i=1}^{n-2} \sum_{j=1}^{i} j = 455}$ **διανύσματα**

Άρα, τα διανύσματα λάθους με 1, 2 ή 3 bits μονάδες είναι συνολικά **575**.

Αναλυτικότερα, συμπληρώνοντας τον πίνακα λαθών μέχρι το μέγεθος $2^{n-k} = 2^{10}$ με διανύσματα λάθους των 4 bits, προκύπτει ο παρακάτω πίνακας:

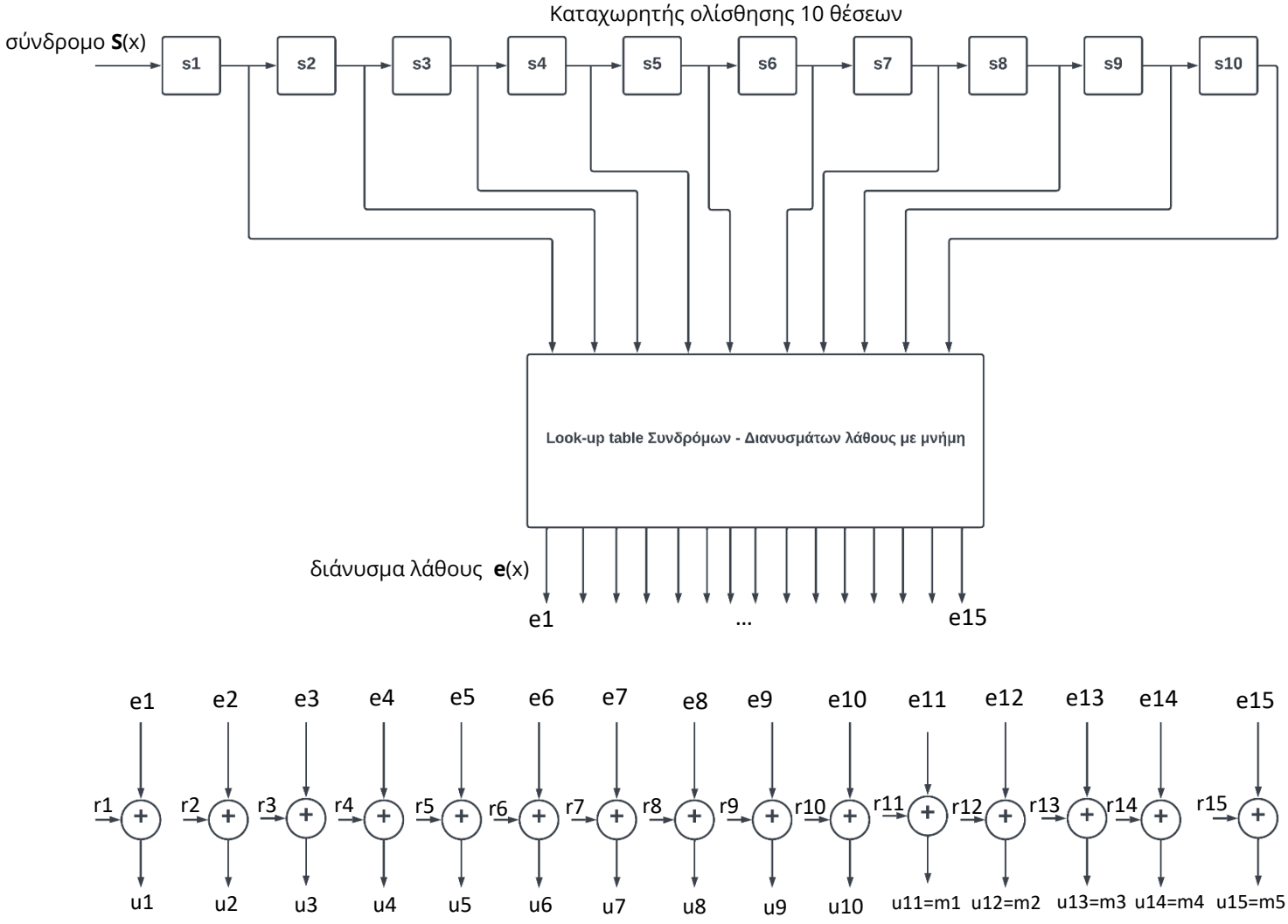
Error vectors (n=15)	
000000000000000	
1 bit set	
100000000000000	15 διανύσματα
010000000000000	
001000000000000	
...	
000000000000001	
2 bits set	
110000000000000	14 διανύσματα
101000000000000	
...	
100000000000001	
011000000000000	13 διανύσματα
...	
010000000000001	
...	...
000000000000011	
3 bits set	
111000000000000	13 διανύσματα
...	
110000000000001	
101100000000000	12 διανύσματα
...	
101000000000001	
...	...
100000000000011	
011100000000000	
...	
000000000000111	
4 bits set	
111100000000000	1024 – 576 = 448 διανύσματα
....	

Πολλαπλασιάζοντας κάθε διάνυσμα λάθους με την $\boldsymbol{H^T}$ προκύπτει το αντίστοιχο σύνδρομο, καθώς $\boldsymbol{s = rH^T = (u + e)H^T = uH^T + eH^T = eH^T}$. Υπάρχει, όμως, η πιθανότητα δύο διανύσματα λάθους να έχουν το ίδιο σύνδρομο ($2^{n=15}$ διανύσματα λάθους αντιστοιχούν σε $2^{n-k=10}$ σύνδρομα). Σε αυτή την περίπτωση, αντιστοιχούμε το σύνδρομο στο διάνυσμα λάθους με το μικρότερο «βάρος» (με τον μικρότερο αριθμό μονάδων), καθώς θεωρούμε ότι ο δίαυλος μας πρόκειται για BSC (Binary Symmetric Channel).

Αφού ολοκληρώσουμε αυτή την διαδικασία, θα έχουμε στην διάθεση μας τον πίνακα συνδρόμων-«μικρού βάρους» διανύσματων λαθών για t=3* διορθώσιμα λάθη, τον οποίο θα χρησιμοποιήσουμε σαν look-up table με μνήμη στο κύκλωμα αποκωδικοποίησης που ακολουθεί.

* πιθανόν να υπάρχουν και σύνδρομα που αντιστοιχούν σε διανύσματα λαθών με «βάρος» 4, αλλά εφόσον δεν συμπεριλαμβάνονται όλα τα διανύσματα με «βάρος» 4 στον πίνακα, ο αριθμός των διορθώσιμων λαθών είναι 3.

Συνολικό κύκλωμα Αποκωδικοποιητή του κώδικα BCH(15,5)



Ερώτημα 3

Η εξομοίωση/υλοποίηση της λειτουργίας Κωδικοποιητή και Αποκωδικοποιητή του BCH(15, 5) του **Ερωτήματος 2** στο MATLAB γίνεται ως εξής:

```
% "Smaller Degree -> MSB" applies to the code below
%% ------------Parameters------------- %%
clear all; close all;
m = 4;
d = primpoly(m); % primitive polynomial
a = gf(2,m,d);
n = 2^m - 1; % length of codeword
t = 3; % correctable errors
g = 1; % generator polynomial

%% -------Generator Polynomial-------- %%
% minimum polynomials
min_poly = gfminpol([1:2*t]', m);

% generator polynomial
% g = ΕΚΠ{ϕ1(x), ϕ3(x), …, ϕ2t-1(x)}
for i = 1:2:(2*t-1)
    g = gfconv(min_poly(i, :), g);
end

k = n-length(g)+1; % length of data-word

%% ---------BCH(n,k) Encoder---------- %%
message_bits = randi([0 1], 1, k); % random binary data stream, m(x)
message = gf(message_bits, m);
x_2t = gf([zeros(1,n-k) 1], m);
shifted_m = conv(x_2t, message); % x^(n-k)*m(x)

[q,b] = deconv(flip(shifted_m), flip(g)); % x^(n-k)*m(x)= q(x)*g(x) + b(x)
transmitted = shifted_m + flip(b); % u(x) = x^(n-k)*m(x) + b(x)

%% ---------Noise Application--------- %%
% change the value and the length of the array below, to check
% if and how many errors can be corrected
% indices must be equal or less than n
errors_indices = [8,12,10,13]; % indices of erroneous bits (starting from MSB)
received = transmitted;

for i=1:length(errors_indices)
    if received(errors_indices(i)) == 1
        received(errors_indices(i)) = 0;
    else
        received(errors_indices(i)) = 1;
    end
end

%% --------Parity Check Matrix-------- %%
% U = [P(k x n-k) | I(k x k)] (systematic encoding)
U = gf([],m,d);
I = eye(k);
for i=1:k
    message_i = gf(I(i,:), m);
    shifted_m = conv(x_2t, message_i);

    [q,b] = deconv(flip(shifted_m), flip(g));
    U(i,:) = shifted_m + flip(b);
end
P = (U(1:k,1:(n-k)));
```

```
P_trans = P';
H = [eye(n-k) P_trans]; % H = [ I(n-k x n-k) | P^T ]
H_trans = H';

%% ------------Error Vectors---------- %%
error_vectors = gf([],m,d);
I = eye(n);

% error vectors with "weight" = 1
for i=1:15
    error_vectors(i,:) = gf(I(i,:),m,d);
end
index = i+1;

% error vectors with "weight" = 2
for i=1:14
   current = error_vectors(i,:);
   for j=i+1:15
       error_vectors(index,:) = current + error_vectors(j,:);
       index = index+1;
   end
end

% error vectors with "weight" = 3
offset = 0;
for i=1:14
   current = error_vectors(i,:);
   offset = offset + 15-i;
   for j=16+offset:120
       error_vectors(index,:) = current + error_vectors(j,:);
       index = index+1;
   end
end

% error vectors with "weight" = 4
offset = 91;
start = 121;
for i=1:14
   current = error_vectors(i,:);
   if i>1
       offset = offset - 15-i;
   end
   start = start + offset;
   for j=start:575
       error_vectors(index,:) = current + error_vectors(j,:);
       if (index == 1024)
           break;
       end
       index = index+1;
   end
   if (index == 1024)
           break;
   end
end

%% ----Syndrome-Error Look-up Table--- %%
syndrome_look_up = [];
for i=1:length(error_vectors)
    s = error_vectors(i,:)*H_trans;
    syndrome_look_up(i,:) = s.x;
end

%% -------Syndrome Calculation-------- %%
syndrome = received*H_trans; % S = rH^T
```

```
%% ---------Error Calculation--------- %%
error = gf(zeros(1,n),m,d); % initialize error

% if syndrome's "weight" is equal or more than 1
% search for the corresponding error vector
if any(syndrome)
    error_found = find(ismember(syndrome_look_up, syndrome.x, 'rows'))';
else
    error_found = [];
end

% in case there are more than 1 corresponding error vectors
% keep the first one (the one with the smaller "weight")
if ~isempty(error_found)
    error = error_vectors(error_found(1),:);
end

%% ----------Error Correction--------- %%
corrected = received + error; % add error vector to received codeword

disp('Data:');
disp(message.x);
disp('Transmitted codeword:');
 disp(transmitted.x);
disp('Received codeword:');
disp(received.x);
disp('Corrected codeword:');
disp(corrected.x);
disp('Error Vector:');
disp(error.x);
disp('Syndrome:');
disp(syndrome.x);
```

Αλλάζοντας τις τιμές και το πλήθος των στοιχείων του πίνακα **errors_indices** στον παραπάνω κώδικα, μπορούμε να αυξήσουμε ή να μειώσουμε τα erroneous bits στην κωδικολέξη που έλαβε ο δέκτης. Στην συνέχεια, παρατηρούμε, μέσω των μηνυμάτων που τυπώνονται, αν έχουν εντοπιστεί και διορθωθεί τα λαθεμένα bits. Για παράδειγμα, τυπώνονται τα εξής, αν έχουμε:

0 erroneous bits

```
Data:
   0   1   1   0   1

Transmitted codeword:
   0   1   1   1   1   0   0   0   1   0   0   1   1   0   1

Received codeword:
   0   1   1   1   1   0   0   0   1   0   0   1   1   0   1

Corrected codeword:
   0   1   1   1   1   0   0   0   1   0   0   1   1   0   1

Error Vector:
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

Syndrome:
   0   0   0   0   0   0   0   0   0   0
```

```
errors_indices = []
```

1 erroneous bit

```
Data:
   1   1   0   1   0

Transmitted codeword:
   1   1   1   1   0   0   0   1   0   0   1   1   0   1   0

Received codeword:
   1   1   1   1   0   0   0   1   0   0   0   1   0   1   0

Corrected codeword:
   1   1   1   1   0   0   0   1   0   0   1   1   0   1   0

Error Vector:
   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0

Syndrome:
   1   1   1   0   1   1   0   0   1   0
```

errors_indices = [11]

```
Data:
   1   0   0   0   0

Transmitted codeword:
   1   1   1   0   1   1   0   0   1   0   1   0   0   0   0

Received codeword:
   1   1   1   1   1   1   0   0   1   0   1   0   0   0   0

Corrected codeword:
   1   1   1   0   1   1   0   0   1   0   1   0   0   0   0

Error Vector:
   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0

Syndrome:
   0   0   0   1   0   0   0   0   0   0
```

errors_indices = [4]

2 erroneous bits

```
Data:
   0   0   1   1   0

Transmitted codeword:
   1   0   1   1   1   1   0   0   0   1   0   0   1   1   0

Received codeword:
   1   0   1   1   1   1   0   1   0   1   0   0   0   1   0

Corrected codeword:
   1   0   1   1   1   1   0   0   0   1   0   0   1   1   0

Error Vector:
   0   0   0   0   0   0   0   1   0   0   0   0   1   0   0

Syndrome:
   1   1   0   1   0   1   1   0   1   0
```

errors_indices = [8,13]

3 erroneous bits

```
Data:
   0   0   1   1   1

Transmitted codeword:
   0   1   1   0   0   1   0   1   0   0   0   0   1   1   1

Received codeword:
   0   0   1   1   0   1   0   1   0   1   0   0   1   1   1

Corrected codeword:
   0   1   1   0   0   1   0   1   0   0   0   0   1   1   1

Error Vector:
   0   1   0   1   0   0   0   0   0   1   0   0   0   0   0

Syndrome:
   0   1   0   1   0   0   0   0   0   1
```

errors_indices = [2,4,10]

4 erroneous bits

```
Data:
   0   1   0   1   0

Transmitted codeword:
   0   0   0   1   1   1   0   1   1   0   0   1   0   1   0

Received codeword:
   1   0   0   0   1   0   0   1   1   1   0   1   0   1   0

Corrected codeword:
   0   0   0   1   1   1   0   1   1   0   0   1   0   1   0

Error Vector:
   1   0   0   1   0   1   0   0   0   1   0   0   0   0   0

Syndrome:
   1   0   0   1   0   1   0   0   0   1
```

errors_indices = [1,4,6,10]

```
Data:
   0   1   1   0   0

Transmitted codeword:
   1   0   1   0   0   0   0   1   1   1   0   1   1   0   0

Received codeword:
   1   1   1   1   0   0   0   1   1   0   0   0   1   0   0

Corrected codeword:
   1   1   0   1   0   1   1   1   1   0   0   0   1   0   0

Error Vector:
   0   0   1   0   0   1   1   0   0   0   0   0   0   0   0

Syndrome:
   0   0   1   0   0   1   1   0   0   0
```

errors_indices = [2,4,10,12]

4< erroneous bits

```
Data:
   1   0   1   0   1

Transmitted codeword:
   1   1   1   0   0   0   1   0   0   1   1   0   1   0   1

Received codeword:
   1   0   0   1   0   0   1   1   0   1   1   0   0   0   1

Corrected codeword:
   1   0   0   1   0   0   1   1   0   1   1   0   0   0   1

Error Vector:
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

Syndrome:
   1   0   1   0   0   1   1   0   1   0
```

```
errors_indices =
  [2,3,4,8,13]
```

```
Data:
   0   1   1   1   1

Transmitted codeword:
   0   0   0   1   0   0   1   1   0   1   0   1   1   1   1

Received codeword:
   0   1   1   0   0   0   1   0   0   1   1   1   1   1   0

Corrected codeword:
   1   1   1   0   0   0   1   0   0   1   1   0   1   0   1

Error Vector:
   1   0   0   0   0   0   0   0   0   0   0   1   0   1   1

Syndrome:
   0   1   0   0   0   1   0   0   1   1
```

```
errors_indices =
  [2,3,4,8,11,15]
```

Από τις παραπάνω εξόδους του προγράμματος μπορούμε να παρατηρήσουμε, ότι για **μέχρι $t = 3$** λαθεμένα bits, ο κώδικας μπορεί να εντοπίσει το διάνυσμα λάθους και να διορθώσει την ληφθείσα κωδικολέξη.

Από την άλλη, για **4** λαθεμένα bits, παρόλο που σε κάποιες περιπτώσεις μπορεί να διορθώσει την κωδικολέξη (1^{ο} παράδειγμα), αυτό δεν ισχύει για όλα τα διανύσματα λάθους «βάρους» 4 (2^{ο} παράδειγμα), όπως παρατηρήσαμε στο **Ερώτημα 2**.

Τέλος, για **πάνω από 4** erroneous bits, ο κώδικας αδυνατεί να εντοπίσει το σωστό διάνυσμα λάθους.