

Politechnika Wrocławska
Wydział Elektroniki

NIEZAWODNOŚĆ I DIAGNOSTYKA
UKŁADÓW CYFROWYCH

SCRAMBLING

Autorzy:

Alicja Myśliwiec 248867

Daria Mileczarska 248894

Dominik Kurowski 248840

Termin zajęć:

Wtorek 11.15-13.00 TN

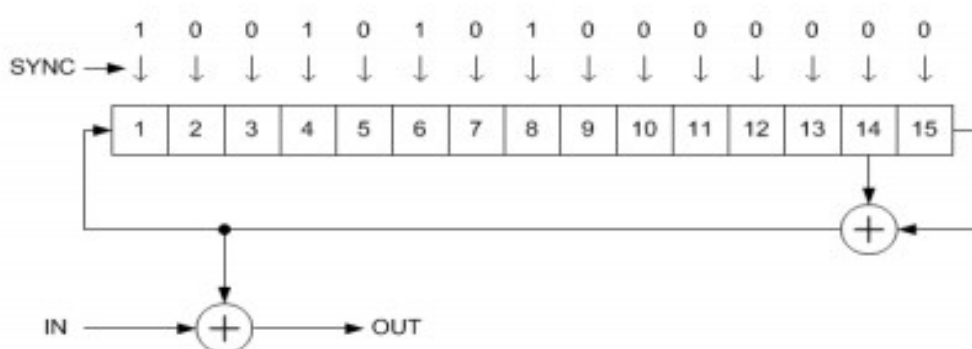
Prowadzący:

dr inż. Jacek Jarnicki

ETAP 1

Czym jest scrambler i descrambler?

Scrambler służy do randomizacji ciągów zer i jedynek. Generowany jest ciąg pseudolosowy, który następnie mieszany jest z danymi wejściowymi. Nie wszystkie ciągi bitów są łatwe do przekazania, szczególnie wymagające są te składające się z samych zer lub samych jedynek. Takim przykładem jest kod NRZ, w którym faza może przyjmować jedną z dwóch wartości przesuniętych względem siebie o 180 stopni reprezentując logiczne 0 i 1. W tym wypadku brak przeźroczystości kodowej wynika z braku synchronizacji odbiornika, która występuje w przypadku długiej sekwencji składających się z samych zer. Metoda scramblingu opiera się na założeniu, że pewne ciągi bitowe są bardziej prawdopodobne niż inne, ale trudniejsze do przesyłania. Zadaniem scramblera jest randomizacja ciągu na łatwiejszy do przekazania. Ciąg ten powstaje przy użyciu bramki XOR, która sumuje kod z pseudolosowymi wartościami. W wyniku takiej operacji powstaje maksymalnie długi ciąg, który jest potem przesyłany kanałem transmisyjnym.



Na rysunku przedstawiony jest schemat scramblera składającego się z dwóch bramek XOR i rejestru przesunego z przykładowym słowem początkowym.

Descrambler dekoduje informacje do postaci pierwotnej. W jednym i drugim urządzeniu używa się rejestrów przesuwanych. Realizacja dekodowania odebranego ciągu bitów w odbiorniku przy pomocy takiego układu pozwala przetestować poprawność działania symulatora, ponieważ ciąg powinien pozostać taki sam

Opis symulacji

```
1 %-----Scrambler-----
2 function [scrambled_data] = Scramble(data, register)
3
4 for i = 1:length(data)
5
6     %Pierwsza bramka xor
7     result = xor(register(length(register)), register(length(register) - 1));
8     %Przesuwanie elementów rejestru
9     for j = 2:length(register)
10         register(length(register) + 2 - j) = register(length(register) + 1 - j);
11     endfor
12     register(1) = result;
13     %Druga bramka xor
14     scrambled_data(i) = xor(result, data(i)) ;
15
16 endfor
17 endfunction
18
```

Powyższa funkcja przedstawia zasady działania scramblera

```
1 %-----Descrambler-----
2 function [descrambled_data] = Descramble (scrambled_data, register)
3
4 for i = 1:length(scrambled_data)
5
6     %Pierwsza bramka xor
7     result = xor(register(length(register)), register(length(register) - 1));
8     %Przesuwanie elementów rejestru
9     for j = 2:length(register)
10         register(length(register) + 2 - j) = register(length(register) + 1 - j);
11     endfor
12     register(1) = result;
13     %Druga bramka xor
14     descrambled_data(i) = xor(result, scrambled_data(i)) ;
15
16 endfor
17 endfunction
18
```

Powyższa funkcja przedstawia zasady działania descramblera

```
1 %-----Funkcja pozwalająca obliczyć długość takich samych sekwencji bitów-----
2 function [bitSequence] = BitSequence (data)
3
4 bitSequence(1) = 1; %tablica, w której zaisujemy kolejne długości ciągów bitów
5 j = 1; %idneks tablicy
6 for i = 2:length(data)
7     if data(i) == data(i-1); %jeżeli bity się powtarzają to zwiększamy wartość w tablicy
8         bitSequence(j)++;
9     else
10         j++; %w przeciwnym wypadku przechodzimy do kolejnego indeksu
11         bitSequence(j) = 1; %przypisujemy wartość początkową
12     endif
13 endfor
14
15 endfunction
16
```

Powyższa funkcja zwraca liczbę takich samych bitów występujących po sobie

Zaprezentowane funkcje zostały wykorzystane do programu, który przekształca ciąg bitów o zadanej długości w inny ciąg o takiej samej długości i został przetestowany dla 3 przypadków.

```

1  %-----Dane początkowe-----
2  register = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]; % inicjalizacja rejestru
3  N = 200; %ilość bitów wejściowych
4
5  %-----Pierwszy przypadek - na wejściu same zera-----
6  data = zeros(1, N);
7
8  scrambled_data = Scramble(data, register);
9  descrambled_data = Descramble(scrambled_data, register);
10 display(scrambled_data);
11 display(descrambled_data);
12
13 histData = BitSequence(scrambled_data);
14 subplot(311)
15 hist(histData,length(histData));
16 xlabel('liczba kolejnych takich samych bitów');
17 ylabel('liczba przypadków');

```

1 przypadek - ciąg samych zer (funkcja zeros())

```

18 %-----Drugi przypadek - na wejściu same jedynki-----
19 data = ones(1,N)
20
21 scrambled_data = Scramble(data, register);
22 descrambled_data = Descramble(scrambled_data, register);
23 display(scrambled_data);
24 display(descrambled_data);
25
26 histData = BitSequence(scrambled_data);
27 subplot(312)
28 hist(histData,length(histData));
29 xlabel('liczba kolejnych takich samych bitów');
30 ylabel('liczba przypadków');

```

2 przypadek - ciąg samych jedynek (funkcja ones())

```

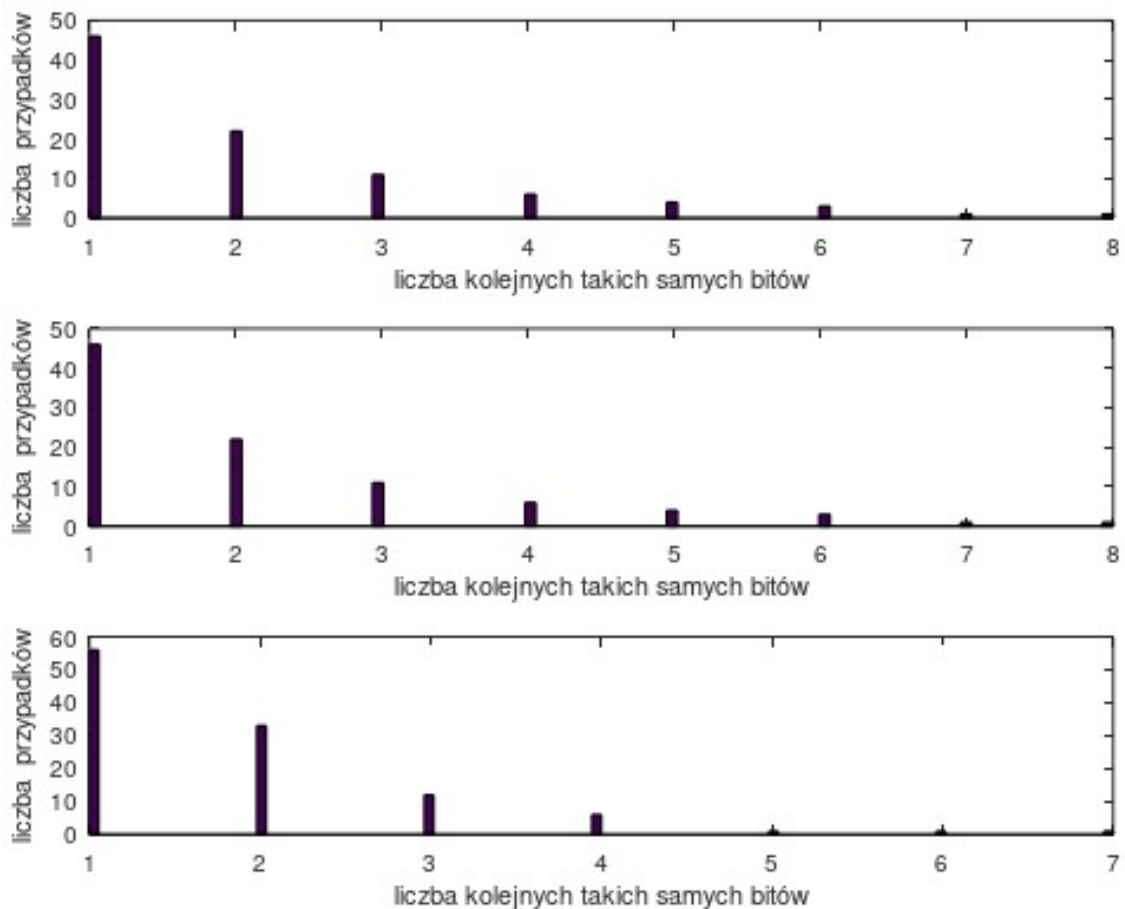
31 %-----Trzeci przypadek - losowy ciąg bitów-----
32 %Generator ciągu bitów o zadanej długości
33 for i = 1:N
34     temp = rand();
35     if temp > 0.5 %Przyjmujemy, że dla wartości powyżej 0.5 bit będzie miał wartość 1
36         data(i) = 1;
37     else %Przyjmujemy, że dla wartości poniżej 0.5 bit będzie miał wartość 0
38         data(i) = 0;
39     end
40 endfor
41
42 scrambled_data = Scramble(data, register);
43 descrambled_data = Descramble(scrambled_data, register);
44 display(data);
45 display(scrambled_data);
46 display(descrambled_data);
47
48 histData = BitSequence(scrambled_data);
49 subplot(313)
50 hist(histData,length(histData));
51 xlabel('liczba kolejnych takich samych bitów');
52 ylabel('liczba przypadków');

```

3 przypadek - losowy ciąg bitów (funkcja rand())

Wykorzystana została funkcja rand(), która zwraca losową wartość z przedziału [0,1]. Ze względu na to, że potrzebujemy wartości całkowitych, w tym wypadku 0 lub 1, założyliśmy, że jeżeli wylosowana liczba przyjmie wartość większą niż pół, uznamy ją za jedynkę, a w przeciwnym wypadku będzie to zero.

Wyniki symulacji

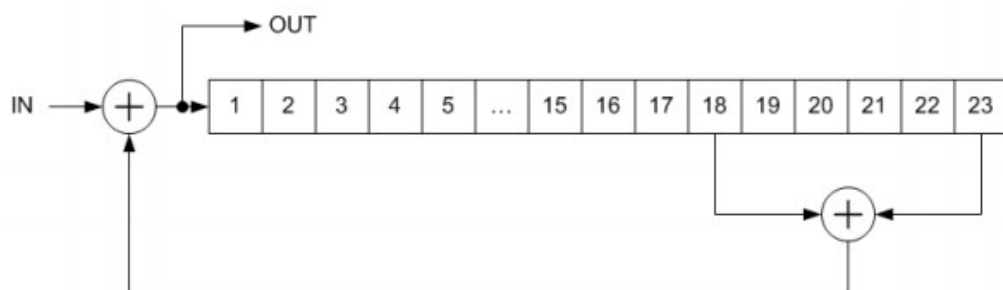


ETAP 2

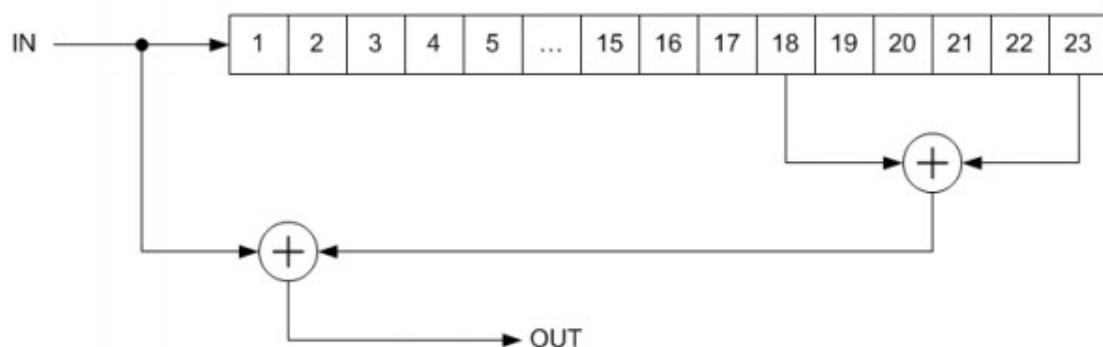
Czym jest standard V.34?

V.34 to standard w komunikacji rekomendowany przez ITU-T, charakteryzujący się pełną transmisją dwukierunkową pomiędzy dwoma modemami wdzwanianymi (połączenie wdzwaniane, ang. *dial-up*) przy przepustowości łącza do 28,8 kbit/s. Inne dodatkowo zdefiniowane przepustowości to: 24,0 kbit/s, 19,2 kbit/s a także przepustowości, które są dopuszczalne przez zalecenia V.32 i V.32bis. Standard ten jest następcą nieoficjalnego standardu V.FC (V.Fast Class) opracowanego przez firmy Hayes i Rockwell International, który był pierwszym szeroko dostępnym protokołem oferującym przepustowość 28,8 kbit/s. Większość modemów standardu V.34 obsługuje także protokół V.FC jakkolwiek nie wszystkie nowoczesne modemy obsługują je oba.

Scrambler i descrambler wg. standardu V.34 to urządzenia różniące się zasadą działania od scramblera i descramblera przedstawionych w etapie pierwszym. Mamy tutaj do czynienia z dwoma różnymi układami. Scrambler i descrambler różnią się od siebie. Poniżej na rysunkach przedstawione zostały schematy danych urządzeń.



Scrambler V.34



Descrambler V.34

Opis symulacji

```
1 %-----Scrambler_DVB-----
2 function [scrambled_data] = Scramble_DVB(data, register)
3
4 for i = 1:length(data)
5
6     %Pierwsza bramka xor
7     result = xor(register(length(register)), register(length(register) - 1));
8     %Przesuwanie elementów rejestru
9     for j = 2:length(register)
10         register(length(register) + 2 - j) = register(length(register) + 1 - j);
11     endfor
12     register(1) = result;
13     %Druga bramka xor
14     scrambled_data(i) = xor(result, data(i)) ;
15
16 endfor
17
18 endfunction
19
```

Powyższa funkcja przedstawia zasady działania scramblera DVB

```
1 %-----Scrambler_V34-----
2 function scrambled_data = Scramble_V34(data, register)
3 for i = 1:length(data)
4     % Wykonaj operację: out = (in XOR (reg[18] XOR reg[23]))
5     result = xor(register(18), register(23));
6     result = xor(data(i), result);
7     % Przesunięcie rejestru w prawo o 1 bit (przy czym ostatni bit w wektorze staje się pierwszym)
8     register = circshift(register, 1);
9     % Po przesunięciu zastąp pierwszy bit rejestru wynikiem operacji
10    register(1) = result;
11    % Zapisz wynik operacji do wektora wyjściowego
12    scrambled_data(i) = result;
13 endfor
14
15 endfunction
16
```

Powyższa funkcja przedstawia zasady działania scramblera V.34

```
1 %-----Descrambler_DVB-----
2 function [descrambled_data] = Descramble_DVB(scrambled_data, register)
3
4 for i = 1:length(scrambled_data)
5
6     %Pierwsza bramka xor
7     result = xor(register(length(register)), register(length(register) - 1));
8     %Przesuwanie elementów rejestru
9     for j = 2:length(register)
10         register(length(register) + 2 - j) = register(length(register) + 1 - j);
11     endfor
12     register(1) = result;
13     %Druga bramka xor
14     descrambled_data(i) = xor(result, scrambled_data(i)) ;
15
16 endfor
17
18 endfunction
19
```

Powyższa funkcja przedstawia zasady działania descramblera DVB

```

1 %-----Descrambler_V34-----%
2 function descrambled_data = Descramble_V34(scrambled_data, register)
3 for i = 1:length(scrambled_data)
4     % Wykonaj operację: out = (in XOR (reg[18] XOR reg[23]))
5     result = xor(register(18), register(23));
6     result = xor(scrambled_data(i), result);
7     % Przesunięcie rejestru w prawo o 1 bit (przy czym ostatni bit w wektorze staje się pierwszym)
8     register = circshift(register, 1);
9     % Po przesunięciu zastąp pierwszy bit rejestru kolejnym bitem wektora wejściowego
10    register(1) = scrambled_data(i);
11    % Zapisz wynik operacji do wektora wyjściowego
12    descrambled_data(i) = result;
13 endfor
14
15 endfunction
16

```

Powyższa funkcja przedstawia zasady działania descramblera V.34

```

1 %-----Funkcja pozwalająca obliczyć długość takich samych sekwencji bitów-----
2 function [bitSequence] = BitSequence (data)
3
4     bitSequence(1) = 1; %tablica, w której zaisujemy kolejne dlugosci cigow bitow
5     j = 1; %idneks tablicy
6     for i = 2:length(data)
7         if data(i) == data(i-1);%jezeli bity sie powtarzaja to zwiekszamy wartosc w tablicy
8             bitSequence(j)++;
9         else
10            j++; %w przeciwnym wypadku przechodzimy do kolejnego indeksu
11            bitSequence(j) = 1; %przypisujemy wartosc poczatkowa
12        endif
13    endfor
14
15 endfunction
16

```

Powyższa funkcja zwraca liczbę takich samych bitów występujących po sobie

```

1 %-----Wytworzenie przeklamania z prawdopodobienstwem P-----
2 function [final_data] = Misrepresentation (original_data, p)
3 for i = 1:length(original_data)
4     temp = rand();
5     if temp <= p % Przyjmujemy, ze dla wartosci mniejszej/rownej od 'p', bit przyjmie wartosc przeciwna
6         final_data(i) = ~original_data(i);
7     else % Przyjmujemy, ze dla wartosci powyzej 'p', wartosc bitu nie ulegnie zmianie
8         final_data(i) = original_data(i);
9     end
10 endfor
11 endfunction
12

```

Powyższa funkcja tworzy przekłamanie bitów z prawdopodobieństwem p

Zaprezentowane funkcje zostały wykorzystane do 3 programów:

- Symulacja bez przekłamań na kanale transmisyjnym
- Symulacja z przekłamaniami na kanale transmisyjnym
- Symulacja z przekłamaniami na pojedynczym bicie

Program 1 - Symulacja bez przekłamań na kanale transmisyjnym

Program symuluje działanie scramblera DVB oraz scramblera V.34 dla trzech przypadków:

- Na wejściu same zera

```
2 %-----Symulacja bez przekłamań na kanale transmisyjnym-----%
3 N = 1000; % Ilość bitów wejściowych
4 register_DVB = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla DVB
5 register_V34 = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla v.34
6 fig1 = figure('Name','Scrambler DVB','NumberTitle','off');
7 fig2 = figure('Name','Scrambler v.34','NumberTitle','off');
8
9 %-----Pierwszy przypadek - na wejściu same zera-----
10 data = zeros(1, N);
11
12 % Scrambler DVB
13 figure(fig1);
14 scrambled_data = Scramble_DVB(data, register_DVB); % Przeprowadź skrablowanie na danych w standardzie DVB
15 descrambled_data = Descramble_DVB(scrambled_data, register_DVB); % Przeprowadź deskrablowanie na danych w standardzie DVB
16 %display(scrambled_data);
17 %display(descrambled_data);
18
19 subplot(311)
20 histData = BitSequence(scrambled_data); % Zamień otrzymane dane na postać umożliwiającą wygenerowanie histogramu
21 hist(histData,length(histData)); % Wygeneruj histogram
22 title('Na wejściu same zera');
23 xlabel('Liczba kolejnych takich samych bitów');
24 ylabel('Liczba przypadków');
25
26 % Scrambler v.34
27 figure(fig2);
28 scrambled_data = Scramble_V34(data, register_V34); % Przeprowadź skrablowanie na danych w standardzie v.34
29 descrambled_data = Descramble_V34(scrambled_data, register_V34); % Przeprowadź deskrablowanie na danych w standardzie v.34
30 %display(scrambled_data);
31 %display(descrambled_data);
32
33 subplot(311)
34 histData = BitSequence(scrambled_data); % Zamień otrzymane dane na postać umożliwiającą wygenerowanie histogramu
35 hist(histData,length(histData)); % Wygeneruj histogram
36 title('Na wejściu same zera');
37 xlabel('Liczba kolejnych takich samych bitów');
38 ylabel('Liczba przypadków');
39
```

- Na wejściu same jedynki

```
40 %-----Drugi przypadek - na wejściu same jedynki-----
41 data = ones(1, N);
42
43 % Scrambler DVB
44 figure(fig1);
45 scrambled_data = Scramble_DVB(data, register_DVB);
46 descrambled_data = Descramble_DVB(scrambled_data, register_DVB);
47 %display(scrambled_data);
48 %display(descrambled_data);
49
50 subplot(312)
51 histData = BitSequence(scrambled_data);
52 hist(histData,length(histData));
53 title('Na wejściu same jedynki');
54 xlabel('Liczba kolejnych takich samych bitów');
55 ylabel('Liczba przypadków');
56
57 % Scrambler v.34
58 figure(fig2);
59 scrambled_data = Scramble_V34(data, register_V34);
60 descrambled_data = Descramble_V34(scrambled_data, register_V34);
61 %display(scrambled_data);
62 %display(descrambled_data);
63
64 subplot(312)
65 histData = BitSequence(scrambled_data);
66 hist(histData,length(histData));
67 title('Na wejściu same jedynki');
68 xlabel('Liczba kolejnych takich samych bitów');
69 ylabel('Liczba przypadków');
70
```

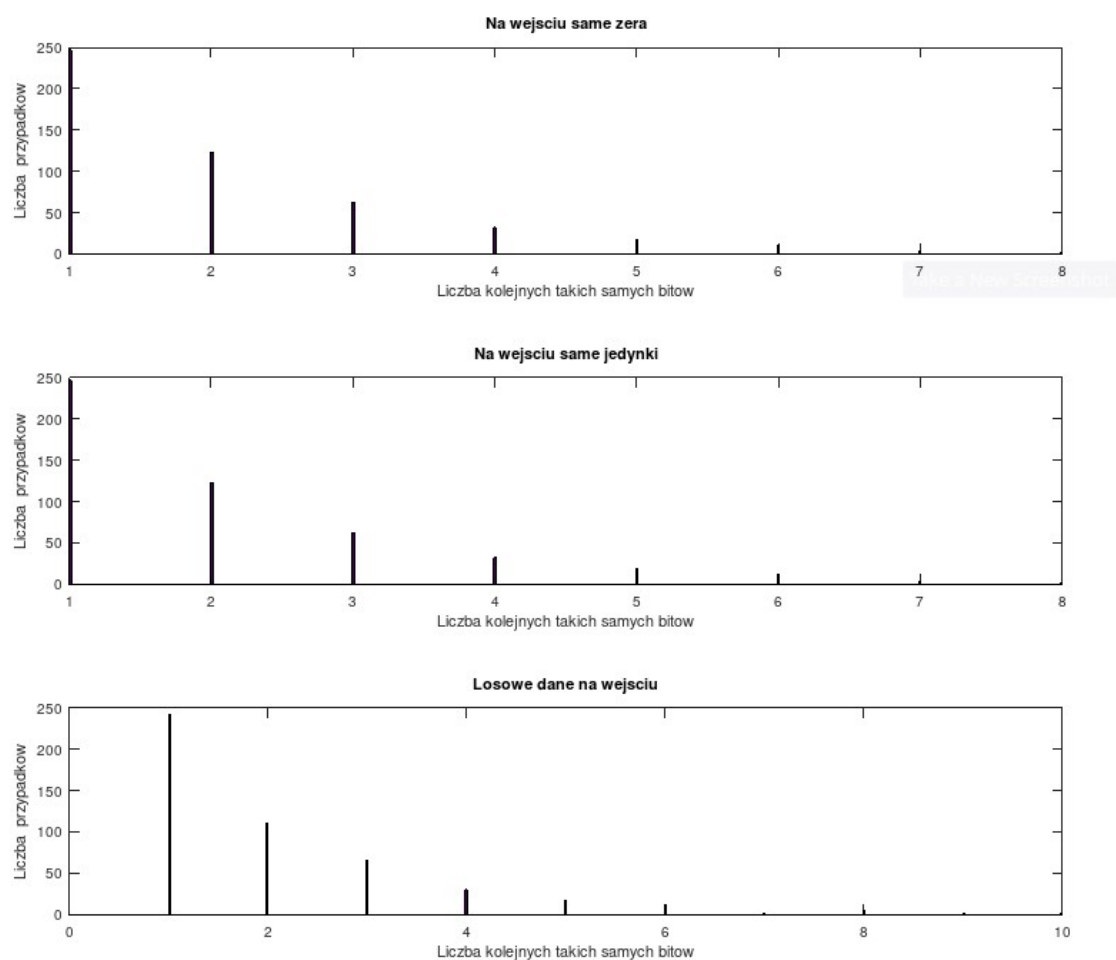
- Losowe dane na wejściu

```

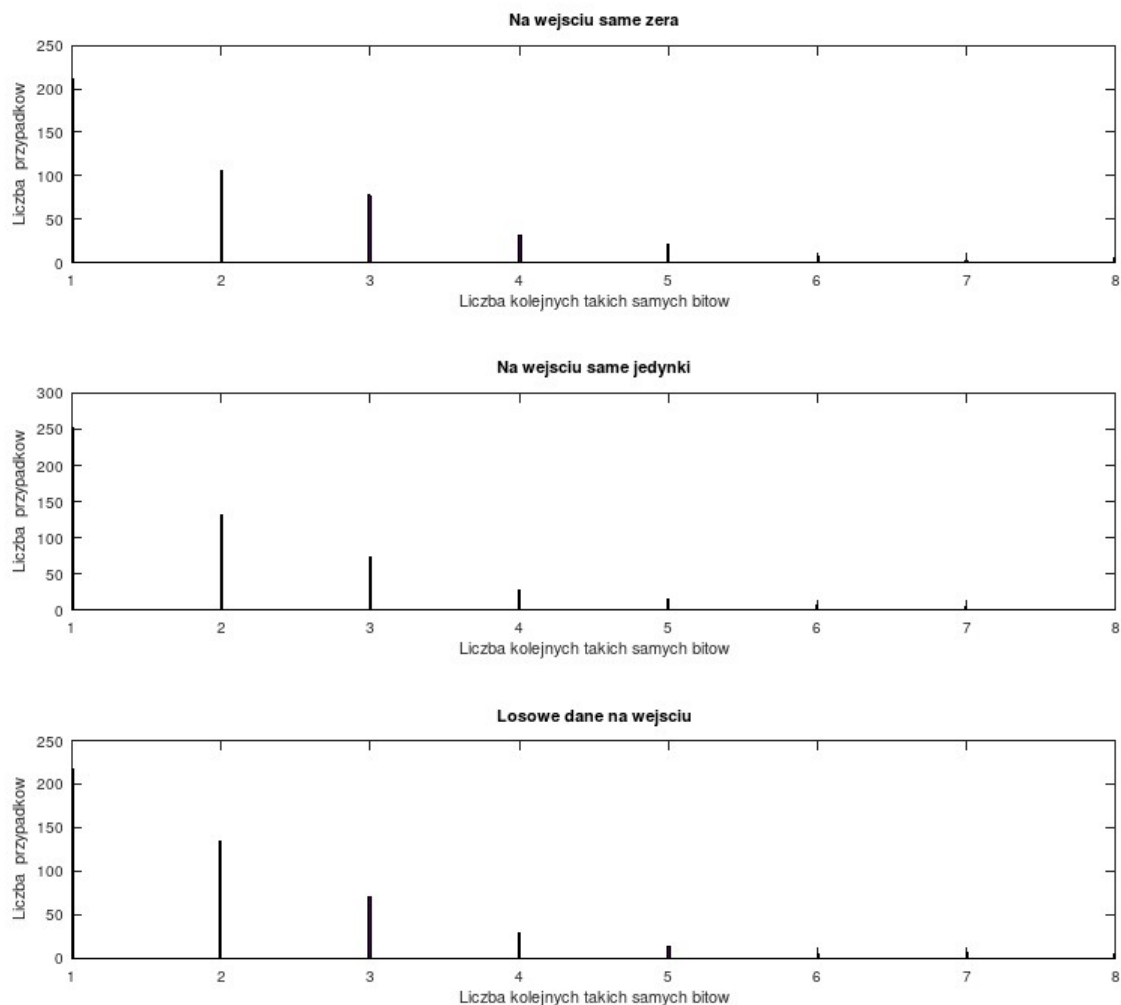
71 %-----Trzeci przypadek - losowe dane na wejściu-----
72 %Generator ciągu bitów o zadanej długości
73 for i = 1:N
74     temp = rand();
75     if temp > 0.5 %Przyjmujemy, że dla wartości powyżej 0.5, bit będzie miał wartość 1
76         data(i) = 1;
77     else %Przyjmujemy, że dla wartości poniżej 0.5, bit będzie miał wartość 0
78         data(i) = 0;
79     end
80 endfor
81
82 % Scrambler DVB
83 figure(fig1);
84 scrambled_data = Scramble_DVB(data, register_DVB);
85 descrambled_data = Descramble_DVB(scrambled_data, register_DVB);
86 %display(scrambled_data);
87 %display(descrambled_data);
88
89 subplot(313)
90 histData = BitSequence(scrambled_data);
91 hist(histData,length(histData));
92 title('Losowe dane na wejściu');
93 xlabel('Liczba kolejnych takich samych bitów');
94 ylabel('Liczba przypadków');
95
96 % Scrambler v.34
97 figure(fig2);
98 scrambled_data = Scramble_V34(data, register_V34);
99 descrambled_data = Descramble_V34(scrambled_data, register_V34);
100 %display(scrambled_data);
101 %display(descrambled_data);
102
103 subplot(313)
104 histData = BitSequence(scrambled_data);
105 hist(histData,length(histData));
106 title('Losowe dane na wejściu');
107 xlabel('Liczba kolejnych takich samych bitów');
108 ylabel('Liczba przypadków');

```

Wyniki symulacji dla programu 1



Powyższe diagramy dotyczą scramblera DVB



Powyższe diagramy dotyczą scramblera V.34

Program 2 - Symulacja z przekłamaniami na kanale transmisyjnym

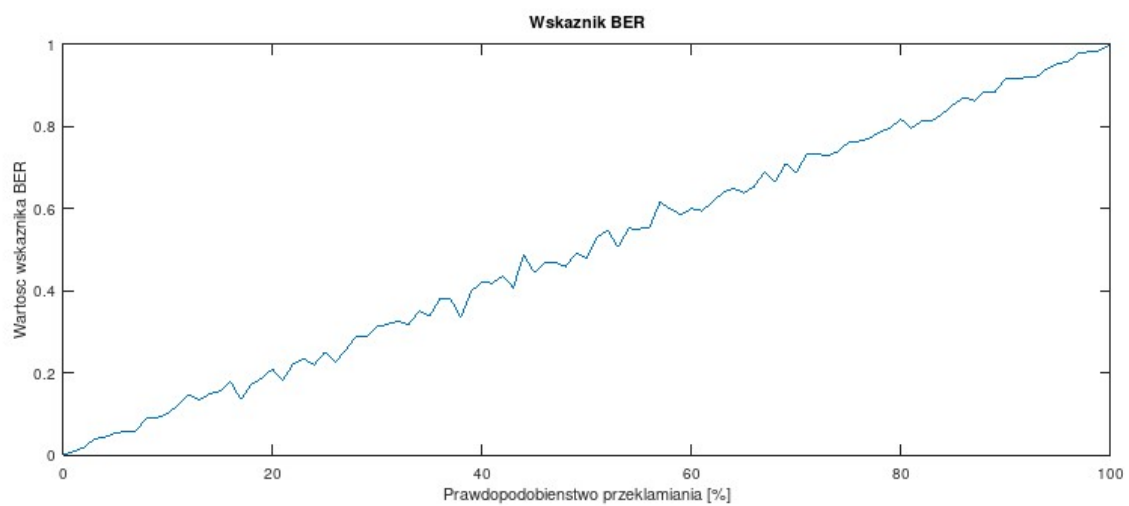
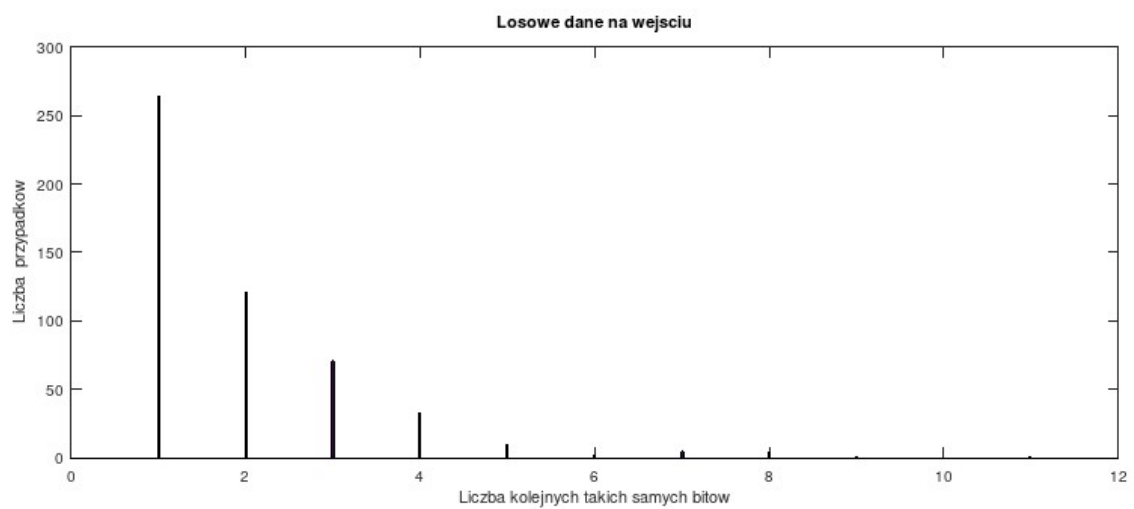
Program symuluje działanie scramblera DVB oraz scramblera V.34 dla losowych danych na wejściu

```

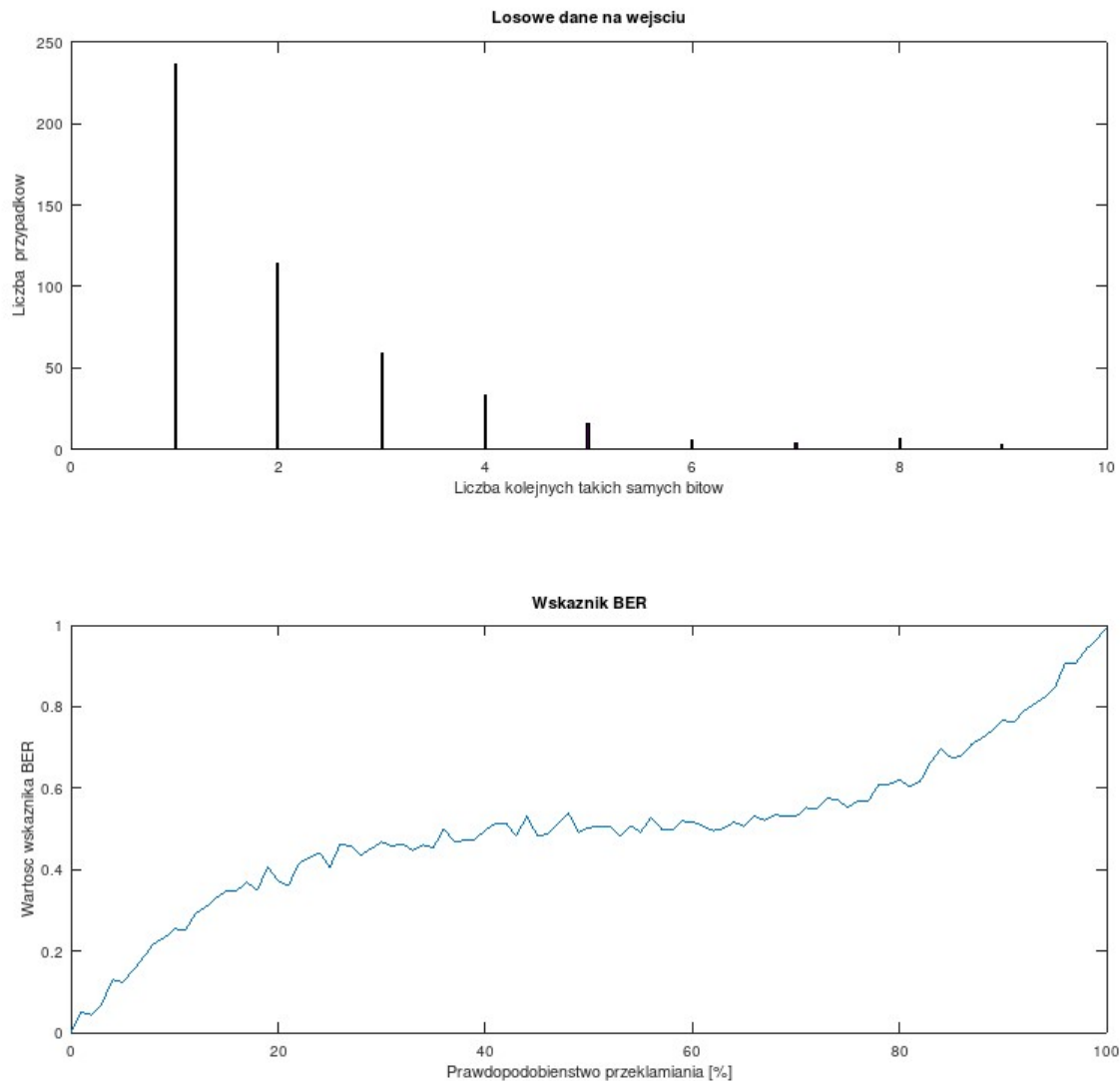
2 %-----Symulacja z przekłamaniami na kanale transmisyjnym-----%
3 N = 1000; % Ilość bitów wejściowych
4 p_range = 0:0.01:1;
5 register_DVB = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla DVB
6 register_V34 = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla v.34
7 fig1 = figure('Name','Scrambler DVB','NumberTitle','off');
8 fig2 = figure('Name','Scrambler v.34','NumberTitle','off');
9
10 %-----Losowe dane na wejściu-----
11 %Generator ciągu bitów o zadanej długości
12 for i = 1:N
13     temp = rand();
14     if temp > 0.5 % Przyjmujemy, że dla wartości powyżej 0.5, bit będzie miał wartość 1
15         data(i) = 1;
16     else % Przyjmujemy, że dla wartości poniżej 0.5, bit będzie miał wartość 0
17         data(i) = 0;
18     end
19 endfor
20
21 % Scrambler DVB
22 figure(fig1);
23 scrambled_data = Scramble_DVB(data, register_DVB); % Przeprowadz skrablowanie na danych w standardzie DVB
24 BER = []; % Utworzyć pusty wektor przechowujący wskaźnik BER dla kolejnych 'p' (prawdopodobieństwo przekłamania bitu)
25 for p = p_range
26     transmitted_data = Misrepresentation(scrambled_data, p); % Zasymuluj wystąpienie przekłaman na kanale transmisyjnym
27     descrambled_data = Descramble_DVB(transmitted_data, register_DVB); % Przeprowadz deskrablowanie danych, które przeszły przez kanał transmisyjnych
28     BER(end+1) = numel(find(descrambled_data~=data))/N; % Oblicz i dodaj na koniec wektora wartość wskaźnika BER
29 endfor
30 BER = cell2mat(BER);
31 display(scrambled_data);
32 display(descrambled_data);
33
34 subplot(211)
35 histData = BitSequence(scrambled_data); % Zamień otrzymane dane na postać umożliwiającą wygenerowanie histogramu
36 hist(histData,length(histData)); % wygeneruj histogram
37 title('Losowe dane na wejściu');
38 xlabel('Liczba kolejnych takich samych bitów');
39 ylabel('Liczba przypadków');
40
41 subplot(212)
42 plot(p_range, BER); % Wygeneruj wykres zależności wartości wskaźnika BER od 'p' - prawdopodobieństwa wystąpienia przekłamania
43 title('Wskaźnik BER');
44 xlabel('Prawdopodobieństwo przekłamania');
45 ylabel('Wartość wskaźnika BER');
46
47 % Scrambler v.34
48 figure(fig2);
49 scrambled_data = Scramble_V34(data, register_V34);
50 BER = [];
51 for p = p_range
52     transmitted_data = Misrepresentation(scrambled_data, p);
53     descrambled_data = Descramble_V34(transmitted_data, register_V34);
54     BER(end+1) = numel(find(descrambled_data~=data))/N;
55 endfor
56 BER = cell2mat(BER);
57 display(scrambled_data);
58 display(descrambled_data);
59
60 subplot(211)
61 histData = BitSequence(scrambled_data);
62 hist(histData,length(histData));
63 title('Losowe dane na wejściu');
64 xlabel('Liczba kolejnych takich samych bitów');
65 ylabel('Liczba przypadków');
66
67 subplot(212)
68 plot(p_range, BER);
69 title('Wskaźnik BER');
70 xlabel('Prawdopodobieństwo przekłamania');
71 ylabel('Wartość wskaźnika BER');
72

```

Wyniki symulacji dla programu 2



Powyższe diagramy dotyczą scramblera DVB



Powyższe diagramy dotyczą scramblera V.34

Program 3 - Symulacja z przekłamaniami na pojedynczym bicie

Program symuluje działanie scramblera DVB oraz scramblera V.34 dla losowych danych na wejściu

```

1 clear;
2 %-----Symulacja z przekłamaniem na pojedynczym bicie-----%
3 N = 1000; % Ilość bitów wejściowych
4 wrong_bit_index = 100;
5 register_DVB = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla DVB
6 register_V34 = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla v.34
7
8 %-----Losowe dane na wejściu-----
9 %Generator ciągu bitów o zadanej długości
10 for i = 1:N
11     temp = rand();
12     if temp > 0.5 %Przyjmujemy, że dla wartości powyżej 0.5, bit będzie miał wartość 1
13         data(i) = 1;
14     else %Przyjmujemy, że dla wartości poniżej 0.5, bit będzie miał wartość 0
15         data(i) = 0;
16     end
17 endfor
18
19 % Scrambler DVB
20 scrambled_data = Scramble_DVB(data, register_DVB); % Przeprowadź skrablowanie na danych w standardzie DVB
21 transmitted_data = scrambled_data;
22 transmitted_data(wrong_bit_index) = !transmitted_data(wrong_bit_index); % Zasymuluj wystąpienie przekłamania na pojedynczym bicie
23 descrambled_data = Descramble_DVB(transmitted_data, register_DVB); % Przeprowadź deskramblowanie danych, które przeszły przez kanał transmisyjnych
24 BER = numel(find(descrambled_data!=data))/N; % Oblicz wartość wskaźnika BER
25 display(BER);
26 display(scrambled_data);
27 display(descrambled_data);
28
29 % Scrambler v.34
30 scrambled_data = Scramble_V34(data, register_V34); % Przeprowadź skrablowanie na danych w standardzie v.34
31 transmitted_data = scrambled_data;
32 transmitted_data(wrong_bit_index) = !transmitted_data(wrong_bit_index); % Zasymuluj wystąpienie przekłamania na pojedynczym bicie
33 descrambled_data = Descramble_V34(transmitted_data, register_V34); % Przeprowadź deskramblowanie danych, które przeszły przez kanał transmisyjnych
34 BER = numel(find(descrambled_data!=data))/N; % Oblicz wartość wskaźnika BER
35 display(BER);
36 display(scrambled_data);
37 display(descrambled_data);
38

```

WNIOSKI:

Przy porównywaniu scramblerów za pomocą ciągu samych jedynek, samych zer oraz losowych wartości, lepiej wypada scrambler wg. standardu V.34, ponieważ ciągi bitów o takiej samej wartości są krótsze, jednak te różnice są nieznaczne. Scrambler DVB tworzy gorszy rozkład jakościowy. Wskaźnik BER = (liczba błędnie odebranych bitów) / (liczba odebranych bitów), zatem powinien być on jak najmniejszy. Jeżeli prawdopodobieństwo przekłamania jest duże, to wskaźnik BER jest generalnie wysoki. Przekłamanie na pojedynczym bicie (bit o indeksie 100 z 1000) daje wskaźnik BER: 0.001 dla DVB, 0.003 dla v.34 (dla programu 3). Przewagę scramblera wg. standardu V.34 widzimy porównując wskaźnik BER. Nawet dla dużego prawdopodobieństwa przekłamania sygnału rzędu 70% nadal utrzymuje wskaźnik BER na poziomie 50%, w przeciwieństwie do scramblera DVB, gdzie wskaźnik BER rośnie liniowo.