Politechnika Wrocławska
Wydział Elektroniki

NIEZAWODNOŚĆ I DIAGNOSTYKA
UKŁADÓW CYFROWYCH

SCRAMBLING

Autorzy:
Alicja Myśliwiec 248867
Daria Milczarska 248894
Dominik Kurowski 248840

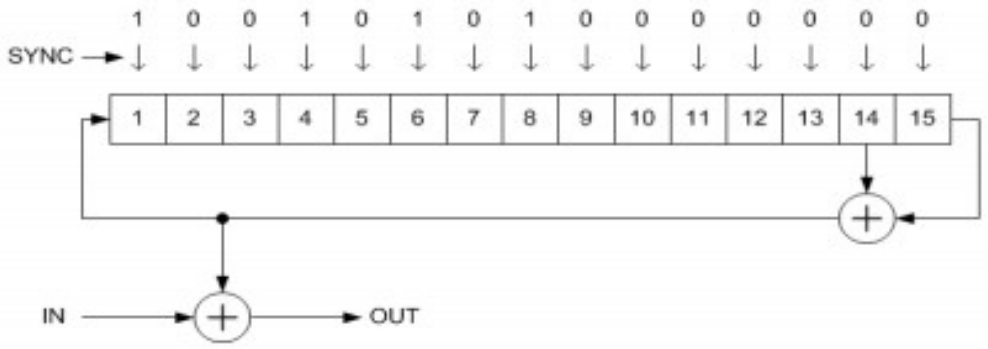
Termin zajęć:
Wtorek 11.15-13.00 TN

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 przesuwnego z przykładowym słowem początkowym.

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

Opis symulacji

```
%---------------------------------Scrambler-------------------------------------
function [scrambled_data] = Scramble(data, register)

  for i = 1:length(data)

        %Pierwsza bramka xor
        result = xor(register(length(register)),register(length(register) - 1));
        %Przesuwanie elementów rejestru
        for j = 2:length(register)
          register(length(register) + 2 - j) = register(length(register) + 1 - j);
        endfor
        register(1) = result;
        %Druga bramka xor
        scrambled_data(i) = xor(result, data(i)) ;

  endfor
endfunction
```

Powyższa funkcja przedstawia zasady działania scramblera

```
%---------------------------------Descrambler-------------------------------------
function [descrambled_data] = Descramble (scrambled_data, register)

  for i = 1:length(scrambled_data)

        %Pierwsza bramka xor
        result = xor(register(length(register)),register(length(register) - 1));
        %Przesuwanie elementów rejestru
        for j = 2:length(register)
          register(length(register) + 2 - j) = register(length(register) + 1 - j);
        endfor
        register(1) = result;
        %Druga bramka xor
        descrambled_data(i) = xor(result, scrambled_data(i)) ;

  endfor
endfunction
```

Powyższa funkcja przedstawia zasady działania descramblera

```
%-------Funckja pozwalająca obliczyć długość takich samych sekwencji bitów------
function [bitSequence] = BitSequence (data)

  bitSequence(1) = 1; %tablica, w ktorej zaisujemy kolejne dlugosci cigow bitow
  j = 1;              %idneks tablicy
  for i = 2:length(data)
    if data(i) == data(i-1);%jezeli bity sie powtarzaja to zwiekszamy wartosc w tablicy
      bitSequence(j)++;
    else
      j++; %w przeciwnym wypadku przechodzimy do kolejnego indeksu
      bitSequence(j) = 1; %przypisujemy wartosc poczatkowa
    endif
  endfor

endfunction
```

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.

```
%----------------------------Dane początkowe-----------------------------------------
register =  [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0];% inicjalizacja rejestru
N = 200; %ilość bitów wejściowych

%----------------------------Pierwszy przypadek - na wejściu same zera----------------
data = zeros(1, N);

scrambled_data = Scramble(data, register);
descrambled_data = Descramble(scrambled_data, register);
display(scrambled_data);
display(descrambled_data);

histData = BitSequence(scrambled_data);
subplot(311)
hist(histData,length(histData));
xlabel('liczba kolejnych takich samych bitów');
ylabel('liczba  przypadków');
```

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

```
%----------------------------Drugi przypadek - na wejściu same jedynki----------------
data = ones(1,N)

scrambled_data = Scramble(data, register);
descrambled_data = Descramble(scrambled_data, register);
display(scrambled_data);
display(descrambled_data);

histData = BitSequence(scrambled_data);
subplot(312)
hist(histData,length(histData));
xlabel('liczba kolejnych takich samych bitów');
ylabel('liczba  przypadków');
```

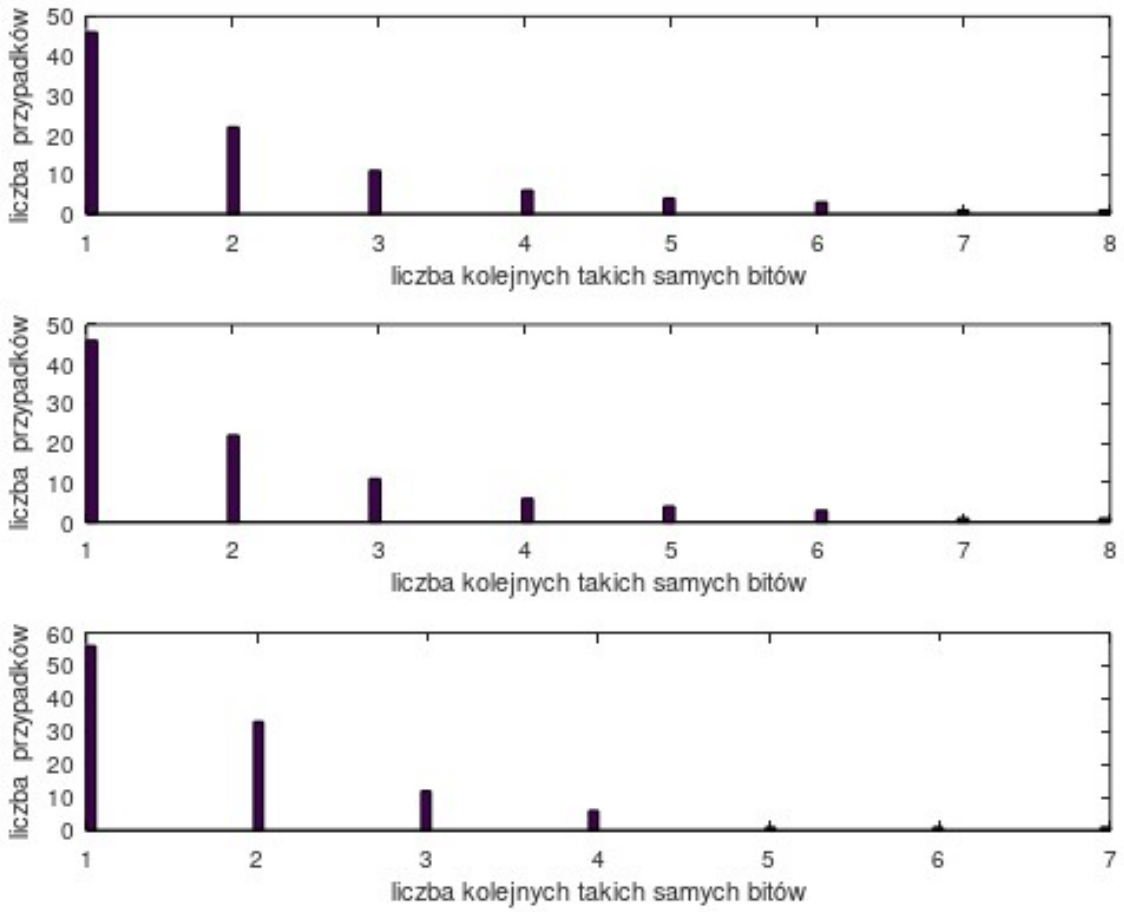
2 przypadek - ciąg samych jedynek (funkcja ones())

```
%----------------------------Trzeci przypadek - losowy ciąg bitów---------------------
%Generator ciągu bitów o zadanej długości
for i = 1:N
  temp = rand();
  if temp > 0.5 %Przyjmujemy, że dla warości powyżej 0.5 bit będzie miał wartość 1
    data(i) = 1;
  else          %Przyjmujemy, że dla warości poniżej 0.5 bit będzie miał wartość 0
    data(i) = 0;
  end
endfor

scrambled_data = Scramble(data, register);
descrambled_data = Descramble(scrambled_data, register);
display(data);
display(scrambled_data);
display(descrambled_data);

histData = BitSequence(scrambled_data);
subplot(313)
hist(histData,length(histData));
xlabel('liczba kolejnych takich samych bitów');
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

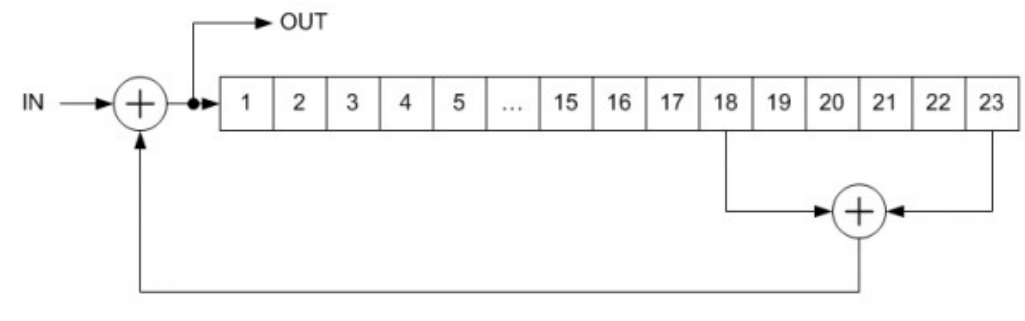


ETAP 2

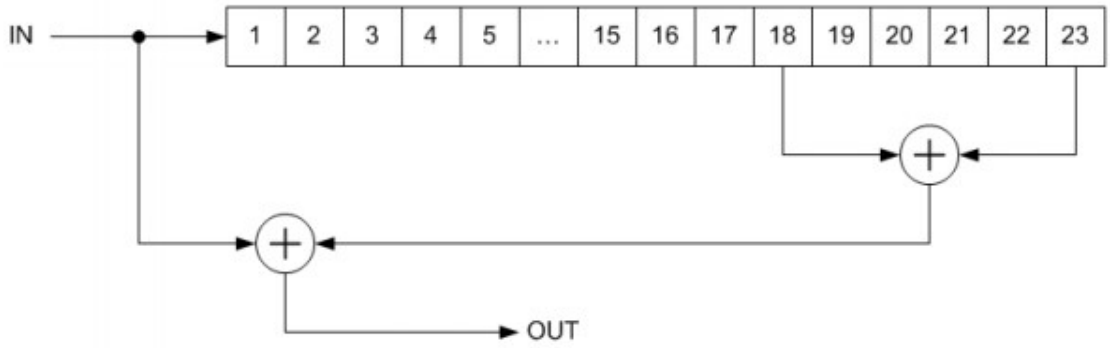
Czym jest standard V.34?

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



Decrambler V.34

Opis symulacji

```
%---------------------------------Scrambler_DVB-------------------------------------
function [scrambled_data] = Scramble_DVB(data, register)

  for i = 1:length(data)

        %Pierwsza bramka xor
        result = xor(register(length(register)),register(length(register) - 1));
        %Przesuwanie elementów rejestru
        for j = 2:length(register)
          register(length(register) + 2 - j) = register(length(register) + 1 - j);
        endfor
        register(1) = result;
        %Druga bramka xor
        scrambled_data(i) = xor(result, data(i)) ;

  endfor

endfunction
```

Powyższa funkcja przedstawia zasady działania scramblera DVB

```
%---------------------------------Scrambler_V34-------------------------------------%
function scrambled_data = Scramble_V34(data, register)
  for i = 1:length(data)
    % Wykonaj operację: out = (in XOR (reg[18] XOR reg[23])
    result = xor(register(18), register(23));
    result = xor(data(i), result);
    % Przesuniecie rejestru w prawo o 1 bit (przy czym ostatni bit w wektorze staje się pierwszym)
    register = circshift(register, 1);
    % Po przesunięciu zastąp pierwszy bit rejestru wynikiem operacji
    register(1) = result;
    % Zapisz wynik operacji do wektora wyjściowego
    scrambled_data(i) = result;
  endfor

endfunction
```

Powyższa funkcja przedstawia zasady działania scramblera V.34

```
%---------------------------------Descrambler_DVB-------------------------------------%
function [descrambled_data] = Descramble_DVB(scrambled_data, register)

  for i = 1:length(scrambled_data)

        %Pierwsza bramka xor
        result = xor(register(length(register)),register(length(register) - 1));
        %Przesuwanie elementów rejestru
        for j = 2:length(register)
          register(length(register) + 2 - j) = register(length(register) + 1 - j);
        endfor
        register(1) = result;
        %Druga bramka xor
        descrambled_data(i) = xor(result, scrambled_data(i)) ;

  endfor
endfunction
```

Powyższa funkcja przedstawia zasady działania descramblera DVB

```
%--------------------------------Descrambler_V34-----------------------------------%
function descrambled_data = Descramble_V34(scrambled_data, register)
  for i = 1:length(scrambled_data)
    % Wykonaj operację: out = (in XOR (reg[18] XOR reg[23])
    result = xor(register(18), register(23));
    result = xor(scrambled_data(i), result);
    % Przesuniecie rejestru w prawo o 1 bit (przy czym ostatni bit w wektorze staje się pierwszym)
    register = circshift(register, 1);
    % Po przesunięciu zastąp pierwszy bit rejestru kolejnym bitem wektora wejściowego
    register(1) = scrambled_data(i);
    % Zapisz wynik operacji do wektora wyjściowego
    descrambled_data(i) = result;
  endfor

endfunction
```

Powyższa funkcja przedstawia zasady działania descramblera V.34

```
%-------Funckja pozwalająca obliczyć długość takich samych sekwencji bitów------
function [bitSequence] = BitSequence (data)

  bitSequence(1) = 1; %tablica, w ktorej zaisujemy kolejne dlugosci cigow bitow
  j = 1;              %idneks tablicy
  for i = 2:length(data)
    if data(i) == data(i-1);%jezeli bity sie powtarzaja to zwiekszamy wartosc w tablicy
      bitSequence(j)++;
    else
      j++; %w przeciwnym wypadku przechodzimy do kolejnego indeksu
      bitSequence(j) = 1; %przypisujemy wartosc poczatkowa
    endif
  endfor

endfunction
```

Powyższa funkcja zwraca liczbę takich samych bitów występujących po sobie

```
%--------------------------------Wytworzenie przekłamania z prawdopodobieństwem P----------------------
function [final_data] = Misrepresentation (orignal_data, p)
  for i = 1:length(orignal_data)
    temp = rand();
    if temp <= p % Przyjmujemy, ze dla wartosci mniejszej/rownej od 'p', bit przyjmie wartosc przeciwna
      final_data(i) = !orignal_data(i);
    else         % Przyjmujemy, ze dla wartosci powyzej 'p', wartosc bitu nie ulegnie zmianie
      final_data(i) = orignal_data(i);
    end
  endfor
endfunction
```

Powyższa funkcja tworzy przekłamania 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

```
%--------------------------Symulacja bez przeklaman na kanale transmisyjnym----------------------%
N = 1000; % Ilosc bitow wejsciowych
register_DVB = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla DVB
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]; % Inicjalizacja rejestru dla v.34
fig1 = figure('Name','Scrambler DVB','NumberTitle','off');
fig2 = figure('Name','Scrambler v.34','NumberTitle','off');

%--------------------------Pierwszy przypadek - na wejsciu same zera-----------------
data = zeros(1, N);

% Scrambler DVB
figure(fig1);
scrambled_data = Scramble_DVB(data, register_DVB); % Przeprowadz skrablowanie na danych w standardzie DVB
descrambled_data = Descramble_DVB(scrambled_data, register_DVB); % Przeprowadz deskrablowanie na danych w standardzie DVB
%display(scrambled_data);
%display(descrambled_data);

subplot(311)
histData = BitSequence(scrambled_data); % Zamien otrzymane dane na postac umozliwiajaca wygenerowanie histogramu
hist(histData,length(histData)); % Wygeneruj histogram
title('Na wejsciu same zera');
xlabel('Liczba kolejnych takich samych bitow');
ylabel('Liczba  przypadkow');

% Scrambler v.34
figure(fig2);
scrambled_data = Scramble_V34(data, register_V34); % Przeprowadz skrablowanie na danych w standardzie v.34
descrambled_data = Descramble_V34(scrambled_data, register_V34); % Przeprowadz deskrablowanie na danych w standardzie v.34
%display(scrambled_data);
%display(descrambled_data);

subplot(311)
histData = BitSequence(scrambled_data); % Zamien otrzymane dane na postac umozliwiajaca wygenerowanie histogramu
hist(histData,length(histData)); % Wygeneruj histogram
title('Na wejsciu same zera');
xlabel('Liczba kolejnych takich samych bitow');
ylabel('Liczba  przypadkow');

```

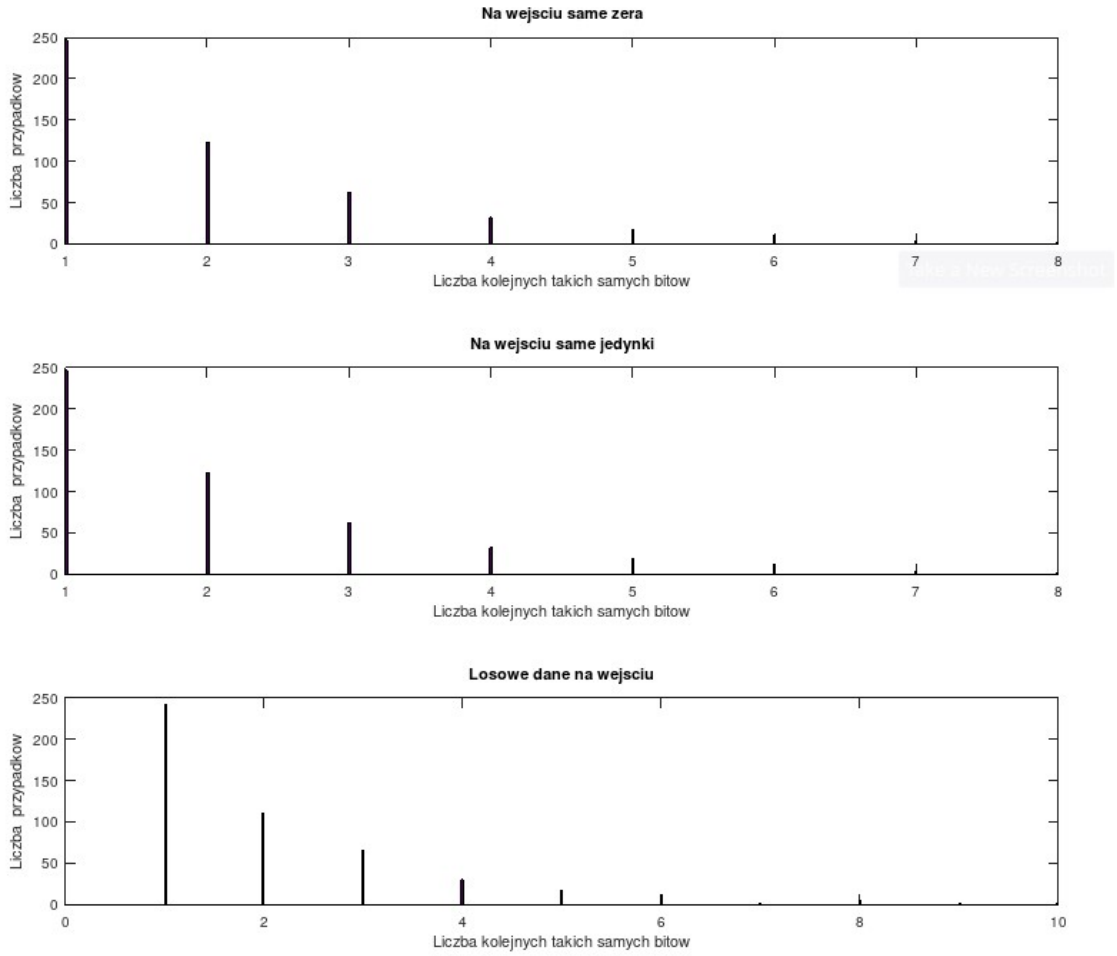
- Na wejściu same jedynki

```
%--------------------------Drugi przypadek - na wejsciu same jedynki-----------------
data = ones(1, N);

% Scrambler DVB
figure(fig1);
scrambled_data = Scramble_DVB(data, register_DVB);
descrambled_data = Descramble_DVB(scrambled_data, register_DVB);
%display(scrambled_data);
%display(descrambled_data);

subplot(312)
histData = BitSequence(scrambled_data);
hist(histData,length(histData));
title('Na wejsciu same jedynki');
xlabel('Liczba kolejnych takich samych bitow');
ylabel('Liczba  przypadkow');

% Scrambler v.34
figure(fig2);
scrambled_data = Scramble_V34(data, register_V34);
descrambled_data = Descramble_V34(scrambled_data, register_V34);
%display(scrambled_data);
%display(descrambled_data);

subplot(312)
histData = BitSequence(scrambled_data);
hist(histData,length(histData));
title('Na wejsciu same jedynki');
xlabel('Liczba kolejnych takich samych bitow');
ylabel('Liczba  przypadkow');

```

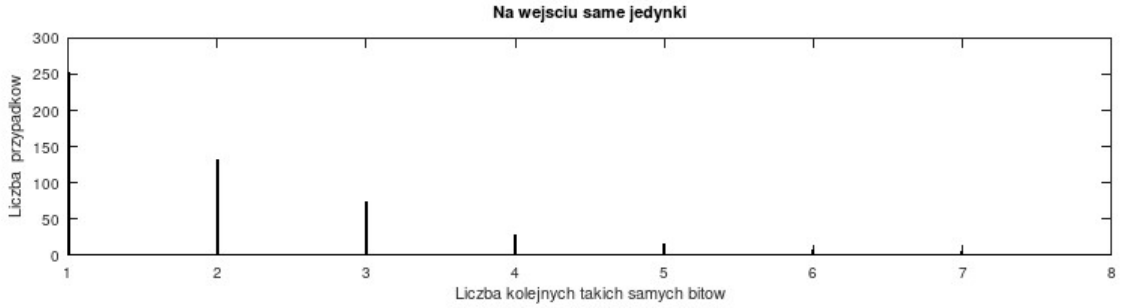
- Losowe dane na wejściu

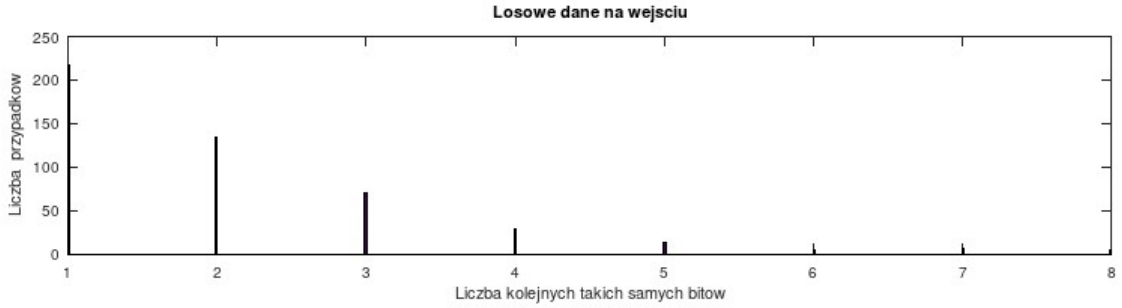
```
71  %-------------------------Trzeci przypadek - losowe dane na wejsciu-----------------
72  %Generator ciagu bitow o zadanej dlugosci
73  for i = 1:N
74    temp = rand();
75    if temp > 0.5 %Przyjmujemy, ze dla wartosci powyzej 0.5, bit bedzie mial wartosc 1
76      data(i) = 1;
77    else          %Przyjmujemy, ze dla wartosci ponizej 0.5, bit bedzie mial wartosc 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 wejsciu');
93  xlabel('Liczba kolejnych takich samych bitow');
94  ylabel('Liczba  przypadkow');
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 wejsciu');
107 xlabel('Liczba kolejnych takich samych bitow');
108 ylabel('Liczba  przypadkow');
```

Wyniki symulacji dla programu 1



Powyższe diagramy dotyczą scramblera DVB





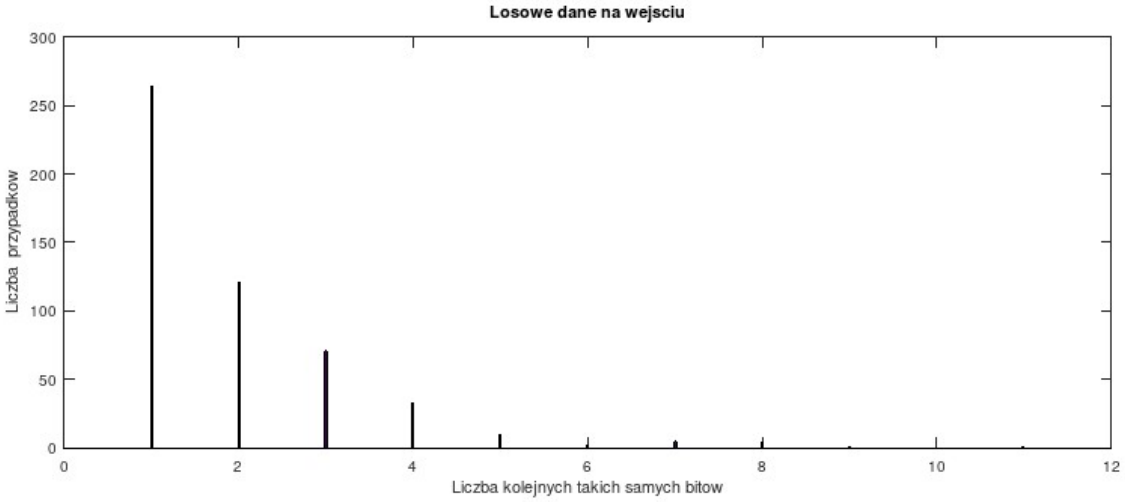
Powyższe diagramy dotyczą scramblera V.34

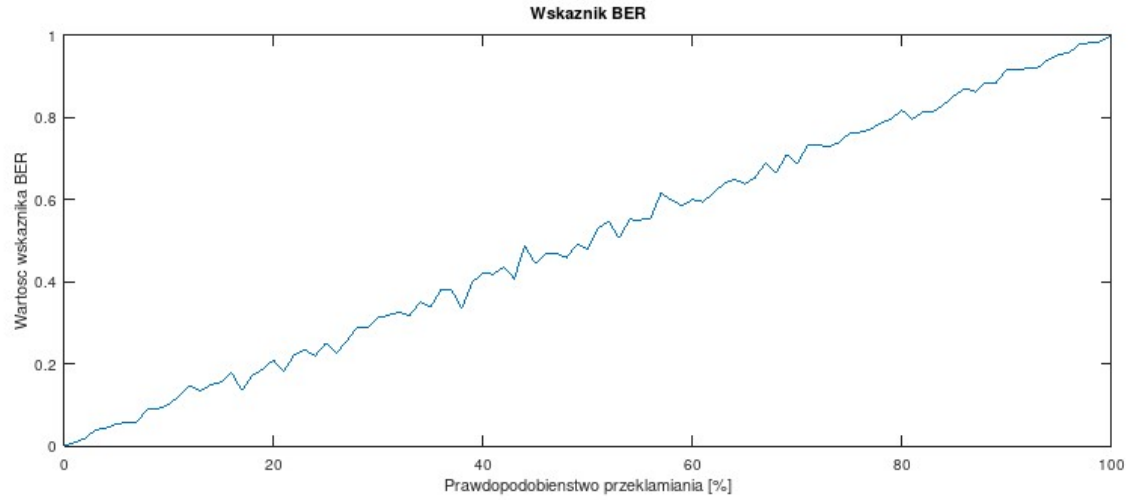
Program 2 - Symulacja z przekłamaniami na kanale transmisyjnym

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

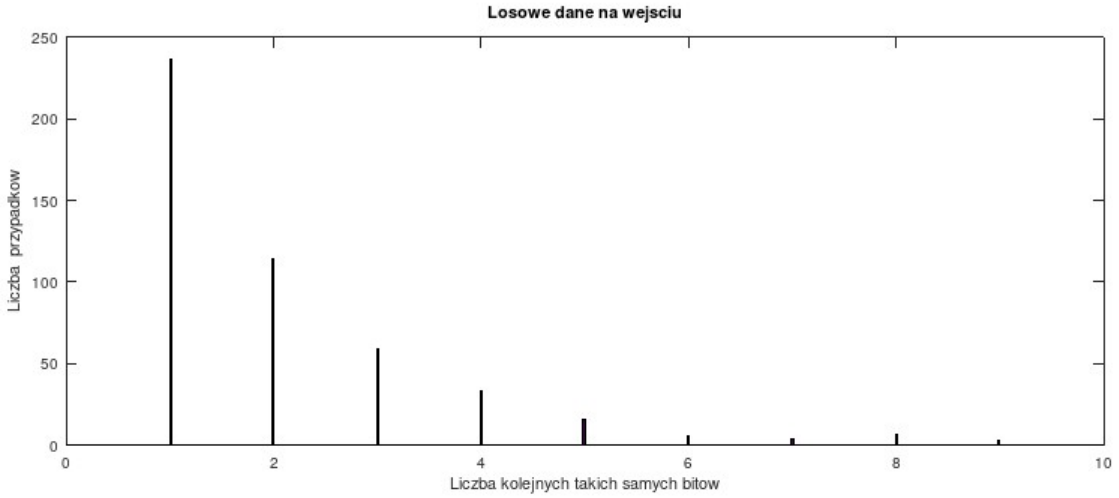
```
%---------------------------Symulacja z przeklamaniami na kanale transmisyjnym----------------------%
N = 1000; % Ilosc bitow wejsciowych
p_range = 0:0.01:1;
register_DVB = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla DVB
register_V34 = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla v.34
fig1 = figure('Name','Scrambler DVB','NumberTitle','off');
fig2 = figure('Name','Scrambler v.34','NumberTitle','off');

%---------------------------Losowe dane na wejsciu-----------------
%Generator ciagu bitow o zadanej dlugosci
for i = 1:N
  temp = rand();
  if temp > 0.5 % Przyjmujemy, ze dla wartosci powyzej 0.5, bit bedzie mial wartosc 1
    data(i) = 1;
  else          % Przyjmujemy, ze dla wartosci ponizej 0.5, bit bedzie mial wartosc 0
    data(i) = 0;
  end
endfor

% Scrambler DVB
figure(fig1);
scrambled_data = Scramble_DVB(data, register_DVB); % Przeprowadz skrablowanie na danych w standardzie DVB
BER = {}; % Utworz pusty wektor przechowujacy wskaznik BER dla kolejnych 'p' (prawdopodobienstwo przeklamania bitu)
for p = p_range
  transmitted_data = Misrepresentation(scrambled_data, p); % Zasymuluj wystapienie przeklaman na kanale transmisyjnym
  descrambled_data = Descramble_DVB(transmitted_data, register_DVB); % Przeprowadz deskramblowanie danych, ktore przeszly przez kanal transmisyjnych
  BER(end+1) = numel(find(descrambled_data!=data))/N; % Oblicz i dodaj na koniec wektora wartosc wskaznika BER
endfor
BER = cell2mat(BER);
display(scrambled_data);
display(descrambled_data);

subplot(211)
histData = BitSequence(scrambled_data); % Zamien otrzymane dane na postac umozliwiajaca wygenerowanie histogramu
hist(histData,length(histData)); % wygeneruj histogram
title('Losowe dane na wejsciu');
xlabel('Liczba kolejnych takich samych bitow');
ylabel('Liczba  przypadkow');

subplot(212)
plot(p_range, BER); % Wygeneruj wykres zaleznosci wartosci wskaznika BER od 'p' - prawdopodobienstwa wystapienia przeklamania
title('Wskaznik BER');
xlabel('Prawdopodobienstwo przeklamiania');
ylabel('Wartosc wskaznika BER');

% Scrambler v.34
figure(fig2);
scrambled_data = Scramble_V34(data, register_V34);
BER = {};
for p = p_range
  transmitted_data = Misrepresentation(scrambled_data, p);
  descrambled_data = Descramble_V34(transmitted_data, register_V34);
  BER(end+1) = numel(find(descrambled_data!=data))/N;
endfor
BER = cell2mat(BER);
display(scrambled_data);
display(descrambled_data);

subplot(211)
histData = BitSequence(scrambled_data);
hist(histData,length(histData));
title('Losowe dane na wejsciu');
xlabel('Liczba kolejnych takich samych bitow');
ylabel('Liczba  przypadkow');

subplot(212)
plot(p_range, BER);
title('Wskaznik BER');
xlabel('Prawdopodobienstwo przeklamiania');
ylabel('Wartosc wskaznika BER');
```

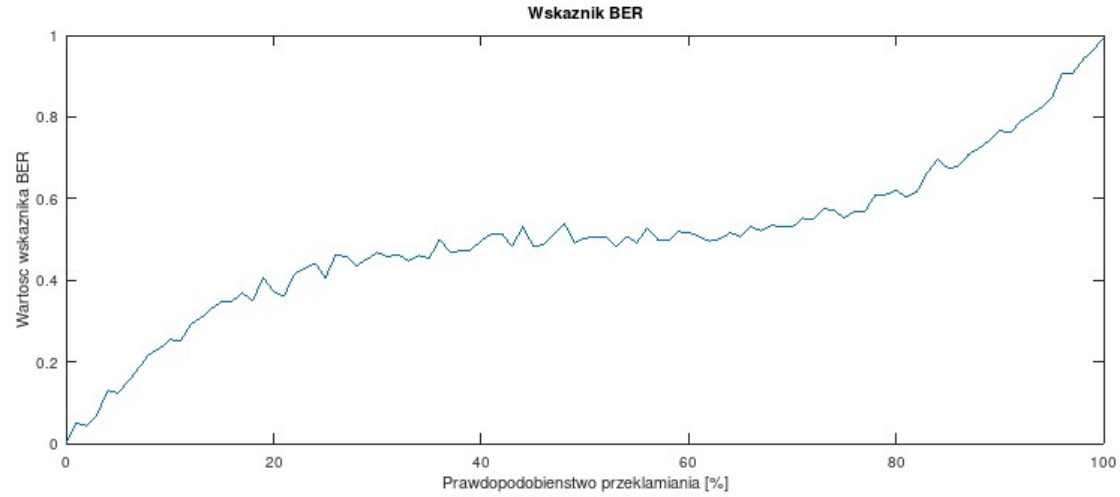
Wyniki symulacji dla programu 2





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

```
clear;
%--------------------------Symulacja z przeklamaniem na pojedynczym bicie----------------------%
N = 1000; % Ilosc bitow wejsciowych
wrong_bit_index = 100;
register_DVB = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla DVB
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]; % Inicjalizacja rejestru dla v.34

%--------------------------Losowe dane na wejsciu----------------
%Generator ciagu bitow o zadanej dlugosci
for i = 1:N
  temp = rand();
  if temp > 0.5 %Przyjmujemy, ze dla wartosci powyzej 0.5, bit bedzie mial wartosc 1
    data(i) = 1;
  else          %Przyjmujemy, ze dla wartosci ponizej 0.5, bit bedzie mial wartosc 0
    data(i) = 0;
  end
endfor

% Scrambler DVB
scrambled_data = Scramble_DVB(data, register_DVB); % Przeprowadz skrablowanie na danych w standardzie DVB
transmitted_data = scrambled_data;
transmitted_data(wrong_bit_index) = !transmitted_data(wrong_bit_index); % Zasymuluj wystapienie przeklamania na pojedynczym bicie
descrambled_data = Descramble_DVB(transmitted_data, register_DVB); % Przeprowadz deskramblowanie danych, ktore przeszly przez kanal transmisyjnych
BER = numel(find(descrambled_data!=data))/N; % Oblicz wartosc wskaznika BER
display(BER);
display(scrambled_data);
display(descrambled_data);

% Scrambler v.34
scrambled_data = Scramble_V34(data, register_V34); % Przeprowadz skrablowanie na danych w standardzie v.34
transmitted_data = scrambled_data;
transmitted_data(wrong_bit_index) = !transmitted_data(wrong_bit_index); % Zasymuluj wystapienie przeklamania na pojedynczym bicie
descrambled_data = Descramble_V34(transmitted_data, register_V34); % Przeprowadz deskramblowanie danych, ktore przeszly przez kanal transmisyjnych
BER = numel(find(descrambled_data!=data))/N; % Oblicz wartosc wskaznika BER
display(BER);
display(scrambled_data);
display(descrambled_data);
```

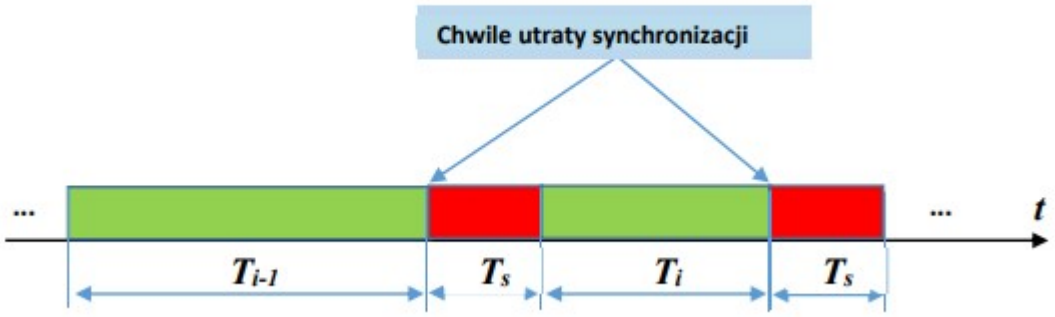
WNIOSKI:

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.

ETAP 3

W ostatnim etapie projektu badaliśmy proces scramblingu oraz jak wpływa on na zjawisko utraty synchronizacji transmisji. W tym celu przyjęliśmy prosty model pozwalający na opisanie procesu utraty i odzyskiwania synchronizacji pomiędzy nadajnikiem i odbiornikiem. System transmisji danych w postaci ciągu bitów składa się z nadajnika, kanału transmisyjnego, w którym z prawdopodobieństwem p mogą występować przekłamania i odbiornika. W nadajniku został zastosowany scrambler, a w odbiorniku odpowiedni descrambler. Odbiornik po odebraniu sekwencji k

kolejnych zer traci synchronizację, co powoduje, że przez czas odpowiadający transmisji l kolejnych bitów nie może prawidłowo odbierać danych. Mechanizm procesu przywrócenia synchronizacji nie jest modelowany w projekcie, jednak powinien następować po czasie odpowiadającym transmisji l bitów.



Rysunek przedstawia naprzemiennie okresy synchronizacji transmisji o losowym czasie trwania (kolor zielony) i okresy przywracania synchronizacja o stałej długości (kolor czerwony). Jakość synchronizacji w takim systemie można scharakteryzować przy pomocy wskaźnika wyrażającego udział czasu pozostawiania systemu w stanie synchronizacji do całego czasu, który upłynął. Postać wskaźnika wygląda następująco:

$$A = \frac{\sum_{i=1}^{i=n} T_i}{n \cdot T_s + \sum_{i=1}^{i=n} T_i}$$

gdzie n jest liczbą przypadków wystąpienia utraty synchronizacji w czasie wykonanej symulacji.

Opis symulacji

Realizacja tego etapu projektu polegała na opracowaniu symulatora pozwalającego na zamodelowanie wyżej opisanego procesu synchronizacji dla trzech przypadków:

- Bez scramblera
- DVB
- V.34

z możliwością zmiany prawdopodobieństwa przekłamania bitu w kanale transmisyjnym

Dla każdego z przypadków należało wyznaczyć zdefiniowany powyżej wskaźnik A i porównać wyniki oraz zbadać jak zmienia się ten wskaźnik w zależności od prawdopodobieństwa p przekłamania bitu w kanale transmisyjnym oraz wartości długości sekwencji zer k i liczby l odpowiadającej czasowi powrotu synchronizacji.

Do wykonania symulacji zostały użyte funkcje stworzone w poprzednich etapach projektu: Scramble_DVB oraz Scramble_V34. Do badania ilości utrat synchronizacji została stworzona i zastosowana funkcja przedstawiona poniżej.

```
%--------------------------------Badanie ilosci utrat synchronizacji-------------------------------------
function synchronized_bits = Synchronization (data, k, I)
  counter = 0; % Licznik wystapien po sobie bitow '0'
  synchronized_bits = 0; % Bity odebrane przy prawidlowej synchronizacji
  i = 1; % Indeks kolejnego bitu danych
  while i < length(data)
    if data(i) == 0 % Sprawdz czy dany bit jest rowny '0'
      counter = counter + 1; % Jezeli tak to inkrementuj licznik wystepujacych po sobie '0'
    else
      counter = 0; % W przeciwnym wypadku zeruj licznik
    endif

    if counter == k % Jezeli licznik jest rowny liczbie bitow zerowcyh potrzebnych do desynchronizacji
      counter = 0; % Zeruj licznik
      i = i + I; % Pomin nastepne I bitow
      continue; % Pomin dalsza czesc petli
    endif

    synchronized_bits = synchronized_bits + 1; % Zwieksz licznik bitow odebranych przy prawidlowej synchronizacji
    i = i + 1; % Przejdz do nastepnego indeksu bitu danych
  endwhile
endfunction

```

Poniżej przedstawiony jest generator bitów o zadanej długości oraz inicjalizacja rejestrów dla scramblerów.

```
clear;
%--------------------------Symulacja bez przeklaman na kanale transmisyjnym----------------------%
N = 16384; % Ilosc bitow wejsciowych
register_DVB = [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]; % Inicjalizacja rejestru dla DVB
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]; % Inicjalizacja rejestru dla v.34
%--------------------------Losowe dane na wejsciu--------------------------%
%Generator ciagu bitow o zadanej dlugosci
for i = 1:N
  temp = rand();
  if temp > 0.5 %Przyjmujemy, ze dla wartosci powyzej 0.5, bit bedzie mial wartosc 1
    data(i) = 1;
  else          %Przyjmujemy, ze dla wartosci ponizej 0.5, bit bedzie mial wartosc 0
    data(i) = 0;
  end
endfor

```

- Zależność A od p

```
%--------------------------Zaleznosc A od p--------------------------%
A1 = A2 = A3 = [];
k = 9; % Dlugosc sekwencji zer
I = 256; % Czas powrotu synchronizacji
p_range = 0:0.05:0.5; % Prawopodobienstwa wystapienia przeklamania
for p = p_range
  % Bez scramblera
  scrambled_data = data;
  transmitted_data = Misrepresentation(scrambled_data, p); % Powstanie zaklaman na kanale transmisyjnym
  synchronized_bits = Synchronization(transmitted_data, k, I); % Zliczanie ilosci desynchronizacji przy odbiorze danych
  A1 = [A1, synchronized_bits / N];

  % Scrambler DVB
  scrambled_data = Scramble_DVB(data, register_DVB);
  transmitted_data = Misrepresentation(scrambled_data, p); % Powstanie zaklaman na kanale transmisyjnym
  synchronized_bits = Synchronization(transmitted_data, k, I); % Zliczanie ilosci desynchronizacji przy odbiorze danych
  A2 = [A2, synchronized_bits / N];

  % Scrambler v.34
  scrambled_data = Scramble_V34(data, register_V34);
  transmitted_data = Misrepresentation(scrambled_data, p); % Powstanie zaklaman na kanale transmisyjnym
  synchronized_bits = Synchronization(transmitted_data, k, I); % Zliczanie ilosci desynchronizacji przy odbiorze danych
  A3 = [A3, synchronized_bits / N];
endfor
p_range = p_range*100;

figure(1)
plot(p_range, A1,'LineWidth', 2)
hold on
plot(p_range, A2,'LineWidth', 2)
plot(p_range, A3,'LineWidth', 2)
hold off

title('Zaleznosc wspolczynnika A od prawdopodobienstwa przeklamania bitu, k = 9, I = 256')
xlabel('Prawdopodobienstwo przeklamania bitu w kanale transmisyjnym [%]')
ylabel('Wspolczynnik pozostania w stanie synchronizacji')
legend({'Bez scramblera', 'Scrambler DVB', 'Scrambler v.34'},'Location','southwest')
legend('boxoff')
axis([0 50])
```

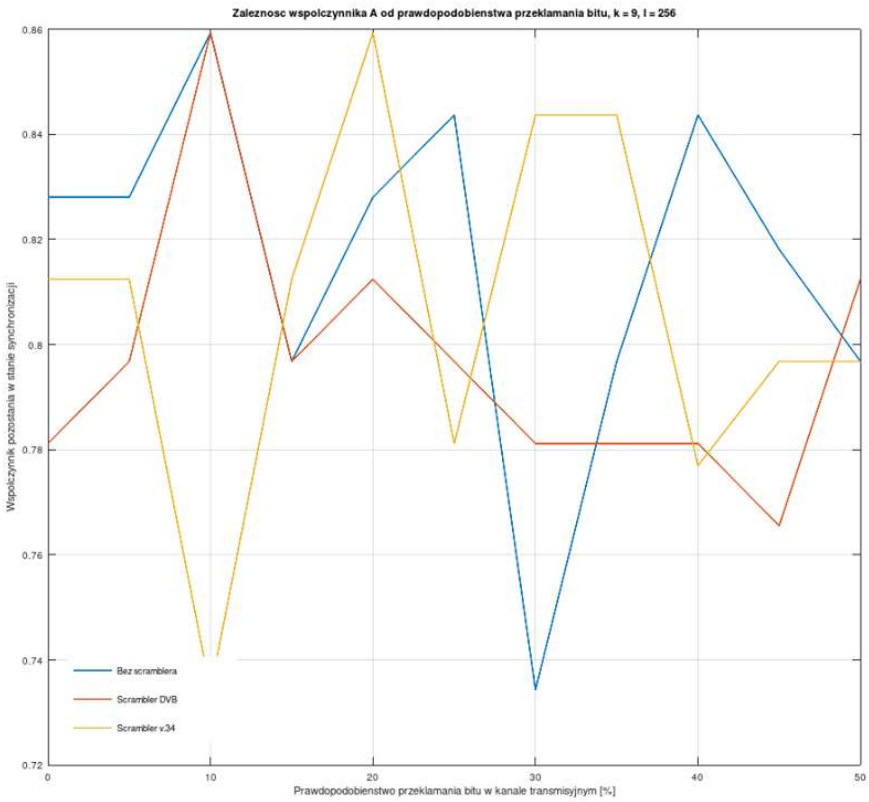
- Zależność A od k

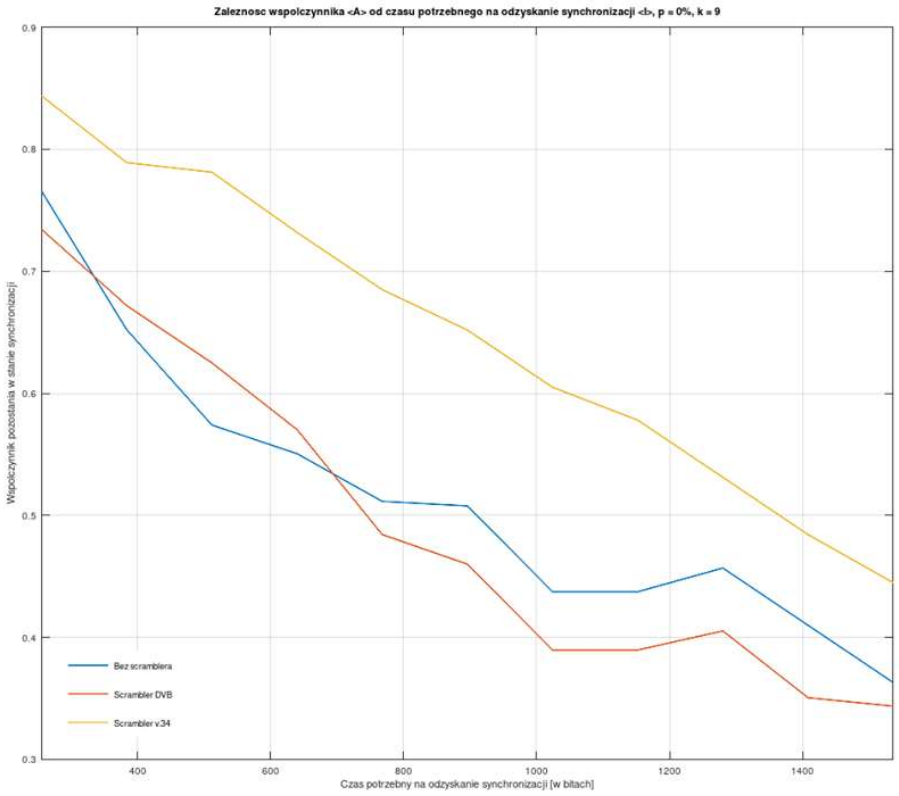
```
%--------------------------Zaleznosc A od k--------------------------%
A1 = A2 = A3 = [];
k_range = [4,5,6,7,8,9,10,11,12,13]; % Dlugosc sekwencji zer
I = 256; % Czas powrotu synchronizacji
p = 0; % Prawopodobienstwo wystapienia przeklamania
for k = k_range
  % Bez scramblera
  scrambled_data = data;
  transmitted_data = Misrepresentation(scrambled_data, p); % Powstanie zaklaman na kanale transmisyjnym
  synchronized_bits = Synchronization(transmitted_data, k, I); % Zliczanie ilosci desynchronizacji przy odbiorze danych
  A1 = [A1, synchronized_bits / N];

  % Scrambler DVB
  scrambled_data = Scramble_DVB(data, register_DVB);
  transmitted_data = Misrepresentation(scrambled_data, p); % Powstanie zaklaman na kanale transmisyjnym
  synchronized_bits = Synchronization(transmitted_data, k, I); % Zliczanie ilosci desynchronizacji przy odbiorze danych
  A2 = [A2, synchronized_bits / N];

  % Scrambler v.34
  scrambled_data = Scramble_V34(data, register_V34);
  transmitted_data = Misrepresentation(scrambled_data, p); % Powstanie zaklaman na kanale transmisyjnym
  synchronized_bits = Synchronization(transmitted_data, k, I); % Zliczanie ilosci desynchronizacji przy odbiorze danych
  A3 = [A3, synchronized_bits / N];
endfor

figure(2)
plot(k_range, A1,'LineWidth', 2)
hold on
plot(k_range, A2,'LineWidth', 2)
plot(k_range, A3,'LineWidth', 2)
hold off

title('Zaleznosc wspolczynnika <A> od ilosci kolejnych zer <k>, p = 0%, I = 256')
xlabel('Ilosc nastepujacych po sobie zer potrzebnych do desynchronizacji')
ylabel('Wspolczynnik pozostania w stanie synchronizacji')
legend({'Bez scramblera', 'Scrambler DVB', 'Scrambler v.34'},'Location','southeast')
legend('boxoff')
axis([4 13])
```

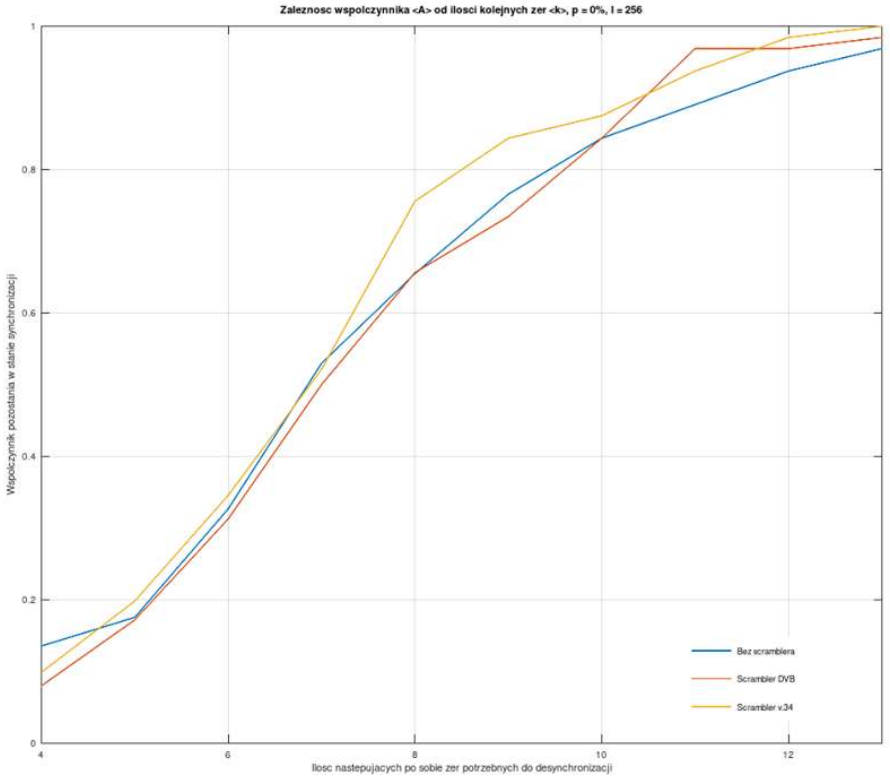
- Zależność A od I

```
%--------------------------Zaleznosc A od I--------------------------%
A1 = A2 = A3 = [];
k = 9; % Dlugosc sekwencji zer
I_range = [256,384,512,640,768,896,1024,1152,1280,1408,1536]; % Czas powrotu synchronizacji
p = 0; % Prawopodobienstwo wystapienia przeklamania
for I = I_range
  % Bez scramblera
  scrambled_data = data;
  transmitted_data = Misrepresentation(scrambled_data, p); % Powstanie zaklaman na kanale transmisyjnym
  synchronized_bits = Synchronization(transmitted_data, k, I); % Zliczanie ilosci desynchronizacji przy odbiorze danych
  A1 = [A1, synchronized_bits / N];

  % Scrambler DVB
  scrambled_data = Scramble_DVB(data, register_DVB);
  transmitted_data = Misrepresentation(scrambled_data, p); % Powstanie zaklaman na kanale transmisyjnym
  synchronized_bits = Synchronization(transmitted_data, k, I); % Zliczanie ilosci desynchronizacji przy odbiorze danych
  A2 = [A2, synchronized_bits / N];

  % Scrambler v.34
  scrambled_data = Scramble_V34(data, register_V34);
  transmitted_data = Misrepresentation(scrambled_data, p); % Powstanie zaklaman na kanale transmisyjnym
  synchronized_bits = Synchronization(transmitted_data, k, I); % Zliczanie ilosci desynchronizacji przy odbiorze danych
  A3 = [A3, synchronized_bits / N];
endfor

figure(3)
plot(I_range, A1,'LineWidth', 2)
hold on
plot(I_range, A2,'LineWidth', 2)
plot(I_range, A3,'LineWidth', 2)
hold off

title('Zaleznosc wspolczynnika <A> od czasu potrzebnego na odzyskanie synchronizacji <I>, p = 0%, k = 9')
xlabel('Czas potrzebny na odzyskanie synchronizacji [w bitach]')
ylabel('Wspolczynnik pozostania w stanie synchronizacji')
legend({'Bez scramblera', 'Scrambler DVB', 'Scrambler v.34'},'Location','southwest')
legend('boxoff')
axis([256 1536])
```

Wyniki symulacji







Otrzymane wyniki jednoznacznie pokazują jak istotny jest rozwój mechanizmów przeciwdziałających skutkom zakłóceń w technologiach telekomunikacyjnych. Sygnał niosący informację ma jakąkolwiek wartość jedynie gdy nie został całkowicie zniekształcony w kanale

transmisyjnym lub gdy to odkształcenie jest odwracalne z poziomu odbiornika. Dzięki standardom takim jak DVB lub v.34 możemy cieszyć się stabilnością podczas korzystania z telewizji bądź internetu mimo nieuniknionych, szkodliwych wpływów zewnętrznych.