Politechnika Wrocławska
Wydział Elektroniki

NIEZAWODNOŚĆ I DIAGNOSTYKA
UKŁADÓW CYFROWYCH

SCRAMBLING

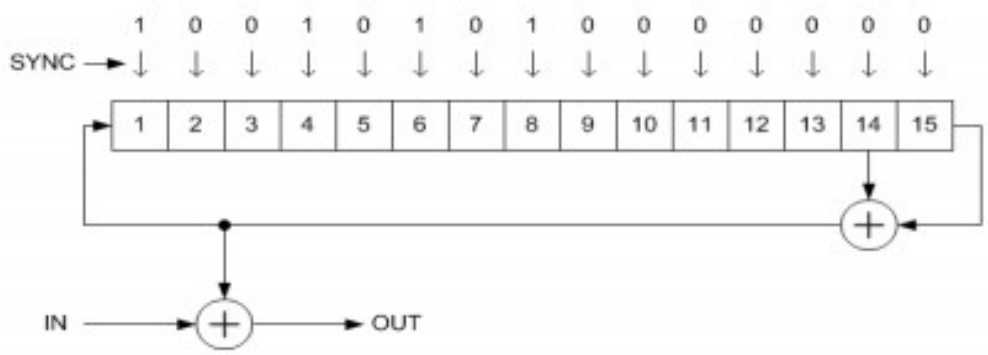
Autorzy:
Alicja Myśliwiec 248867
Daria Milczarska 248894
Dominik Kurowski 248840

Termin zajęć:
Wtorek 11.15-13.00 TN

Prowadzący:
dr inż. Jacek Jarnicki

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