POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka

Niezawodność i Diagnostyka Układów Cyfrowych 2 - Projekt

Transmisja w systemie ARQ
(Automatic Repeat Request)

AUTORZY:
Konrad Piechota (235968)
Łukasz Szumilas (236068)

PROWADZĄCY:
Dr inż. Jacek Jarnicki

WROCŁAW 2018

Spis treści

1. Wstęp **2**

2. Systemy transmisyjne ARQ **3**
2.1. Zastosowanie. 3
2.2. Rozwiązania sprzętowo – programowe 3

3. Opis symulatora **4**
3.1. Założenia. 4
3.2. Algorytmy, opis programów. 4

4. Eksperyment symulacyjny **7**
4.1. Bit parzystości bez kanału zwrotnego. 7
4.2. Bit parzystości z kanałem zwrotnym. 8
4.3. System CRC. 9
4.4. Zadanie projektowe. 9

5. Wnioski **11**

Bibliografia **12**

Programy **13**

Rozdział 1

Wstęp

Grupa projektowa z Niezawodności i Diagnostyki Układów Cyfrowych miała za zadanie poznać metody komputerowego modelowania losowości. Opierając się na tych metodach trzeba było, używając środowiska Matlab, napisać oprogramowanie umożliwiające symulację systemu transmisji danych cyfrowych ARQ (Automatic Repeat Request) w postaci cyfrowej.

Dzięki wybranym metodom analizy statystycznej, po symulacji należało zoptymalizować parametry systemu. W przykładzie systemu ARQ należało wyznaczyć taki kod nadmiarowy (używany w kanale przepływu informacji), by poziom zakłamania bitów i przymus ponownego przesłania pakietu danych ze względu na te błędy był jak najniższy.

Rozdział 2

Systemy transmisyjne ARQ

Przy przesyle danych w kanale transmisyjnych może dojść do zakłamania bitów informacji. W rzeczywistym świecie ciężko znaleźć kanał, który będzie przekazywał dane w 100 % niezmienione. Ważne jest więc sprawdzanie przesyłanych pakietów, by w razie błędów nie dotarły do odbiorcy w takim stanie. Jednym ze sposobów sprawdzania danych jest system transmisyjny ARQ (Automatic Repeat Request).

System ten polega na dopisaniu do określonego pakietu informacji dodatkowych bitów, które w jednoznaczny sposób będą potrafiły go zidentyfikować wśród wszystkich możliwych ciągów bitów. Po stronie odbiorcy, w dekoderze, dodatkowe bity są sprawdzane i w przypadku wystąpienia błędów następuje prośba o ponowne przesłanie danego pakietu.

2.1 Zastosowanie

System ARQ znajduje zastosowanie w protokole sterowania transmisją TCP w modelu OSI, czyli standardowej strukturze komunikacji sieciowej [1]. Dzięki niemu warstwa TCP upewnia się o prawidłowości przesyłu danych.

Stosuje się go także w telekomunikacji. Sektor Normalizacji Telekomunikacji ITU posiada standard używany w lokalnych sieciach komputerowych, w którym system ARQ pozwala przechwycić złe dane w zaszumionych sygnale. W przeszłości był używany podczas transmisji danych przy pomocy fal krótkich, np. w telegramach.

2.2 Rozwiązania sprzętowo – programowe

Istnieją trzy warianty systemu ARQ: stop-and-wait (SAW), go-back-N (GBN) oraz selective repeat (SR) [2]. W metodzie SAW, nadawca czeka na informacje od odbiorcy o wiarygodności przesłanych danych. Jeśli odbiorca otrzymał dobry pakiet, nadawca otrzymuje tzw. „Acknowledgement” (ACK) i może przesyłać kolejny pakiet. Jeśli zostanie przesłana negatywna odpowiedź „NotAcknoledgement” (NACK) – oznacza to błąd i pakiet jest przesyłany ponownie. Metoda ta jest często wykorzystywana w komunikacji, gdzie naraz wysyłać lub odbierać pakiet może tylko jedna ze stron. W metodzie GBN nadawca wysyła pakiety cały czas, gdzie kanał zwrotny traktuje tylko jako odbiornik dla potwierdzeń. Nadawca wysyła ponownie pakiet błędny oraz wszystkie kolejne. W ostatniej metodzie SR, gdzie transmisja również odbywa się w sposób ciągły, z tą różnicą, że otrzymanie wiadomości o błędnym pakiecie powoduje ponowne wysłanie tylko tego jednego. W każdym z wymienionych powyżej systemów jako kod nadmiarowy wykorzystuje się kod cykliczny CRC. Każda oddzielna paczka informacji jest dzielona przez dany z góry wielomian. Kodem nadmiarowym jest reszta z dzielenia danej paczki przez ten wielomian.

Rozdział 3

Opis symulatora

By sprawdzić możliwości systemu ARQ, przyjrzeć się jego konstrukcji i sposobowi działania, grupa musiała zaprojektować jego teoretyczny model.

3.1 Założenia

Na model symulatora składał się:
- generator: miał za zadanie przedstawiać informację przekształconą w ciąg bitów. Ilość tych bitów miała być na tyle duża, żeby wykresy dotyczące statystyk przesłanych bitów były stabilne. Grupa przyjęła ilość równą 99 936 bitów. Ta dokładna liczba to wspólna wielokrotność pakietów danych 4, 8, 16, 32 i 48 bitowych najbliższa 100 tysiącom. Chodziło o to, by bity przesyłane były w pełnych pakietach. Generowanie informacji przebiegało losowo za pomocą funkcji rand.

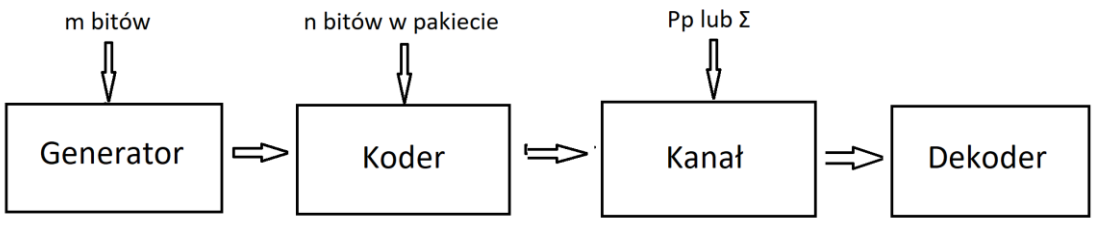
- koder: tutaj, już po podzieleniu informacji na pakiety, dodawany był kod nadmiarowy sprawdzający poprawność przesyłu danych. Ilość bitów w jednym pakiecie była różnorodna, pozwalając na porównanie wpływu długości pakietu na dobry przesył informacji.

- kanał: symulował zakłócenia mające swoje odniesienie w rzeczywistości, gdzie informacja podczas transmisji poddana jest różnym czynnikom wpływającym na dokładność danych. Przekłamanie opierało się na a) coraz to większym prawdopodobieństwie (**Pp**), z którym bit może zmienić swoją wartość z 0 na 1 lub na odwrót lub b) dodaniem do bitu szumu białego (**Σ**), którego intensywność jest teoretycznie statyczna dla całej informacji, mimo to może ją przekłamać.

- dekoder: sprawdzał kod nadmiarowy, porównując go z tym w koderze i liczył błędne pakiety.

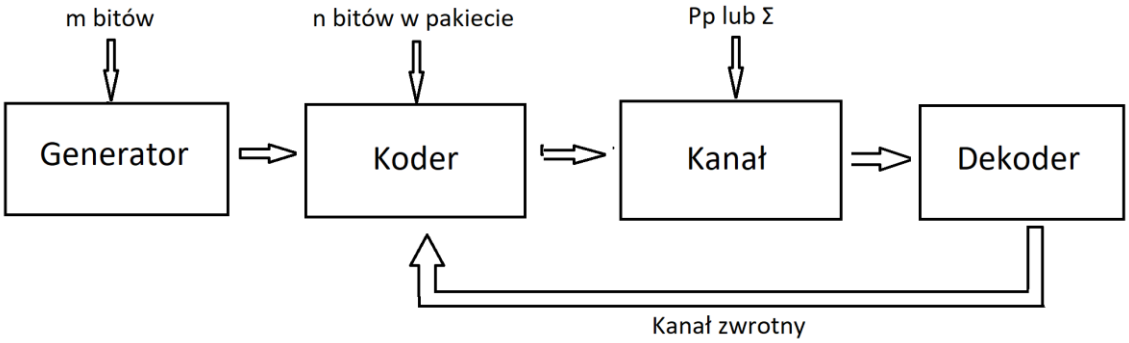
3.2 Algorytmy, opis programów

Podczas pierwszej części eksperymentu trzeba było zasymulować proste przesyłanie danych przez kanał transmisyjny z zakłóceniami, bez kanału zwrotnego (Rys. 3.1).



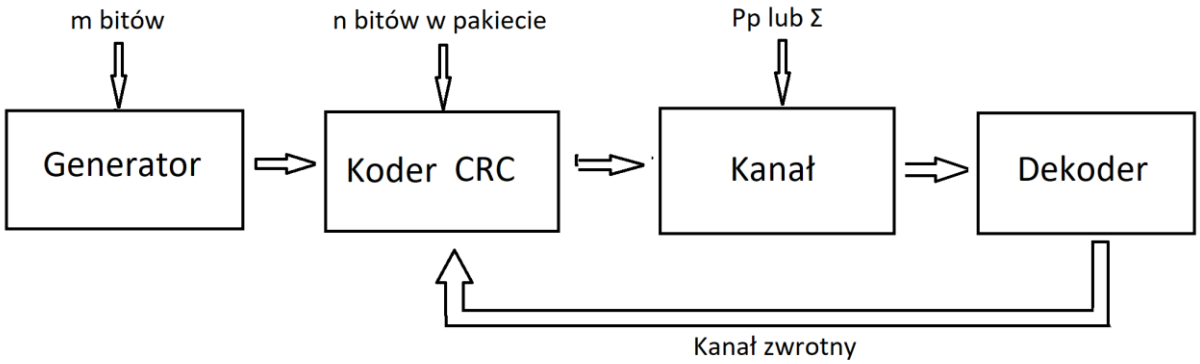
Rys. 3.1 Algorytm kanału z bitem parzystości bez kanału zwrotnego

Kolejnym programem, już trudniejszym do zrealizowania, był ten z kanałem zwrotnym (Rys. 3.2). Prócz ponownego wysłania błędnych informacji ważnym było zapamiętanie oryginalnego ciągu bitów, by móc go później porównać z tym przepuszczonym przez kanał.



Rys. 3.2 Algorytm kanału z bitem parzystości z kanałem zwrotnym

Po zrealizowaniu prostego kodu nadmiarowego jakim był bit parzystości nadszedł czas na kod CRC (Rys. 3.3). Przy realizacji algorytmu różnił się tylko sposobem sprawdzania błędnej informacji.



Rys. 3.3 Algorytm kanału z kodem CRC z kanałem zwrotnym

Przykład działania kodu CRC [3]:

1)

```
11010011101110 000 <--- 14 bitów danych + 3 wyzerowane bity
1011               <--- 4-bitowy dzielnik CRC
01100011101110 000 <--- wynik operacji XOR
 1011
00111011101110 000
  1011
00010111101110 000
   1011
00000001101110 000
       1011
00000000110110 000
        1011
00000000011010 000
         1011
00000000001100 000
          1011
00000000000111 000
           101 1
00000000000010 100
            10 11
------------------
00000000000000 010 <--- CRC
```

2)

```
11010011101110 010 <--- przesłany bez przekłamań ciąg 14 bitów danych + CRC
1011               <--- ustalony uprzednio, 4-bitowy dzielnik
01100011101110 010 <--- wynik operacji XOR
 1011
00111011101110 010
  1011
00010111101110 010
   1011
00000001101110 010
       1011
00000000110110 010
        1011
00000000011010 010
         1011
00000000001100 010
          1011
00000000000111 010
           101 1
00000000000010 110
            10 11
------------------
00000000000000 000 <--- wynik operacji równy 0 oznacza poprawną transmisję
```

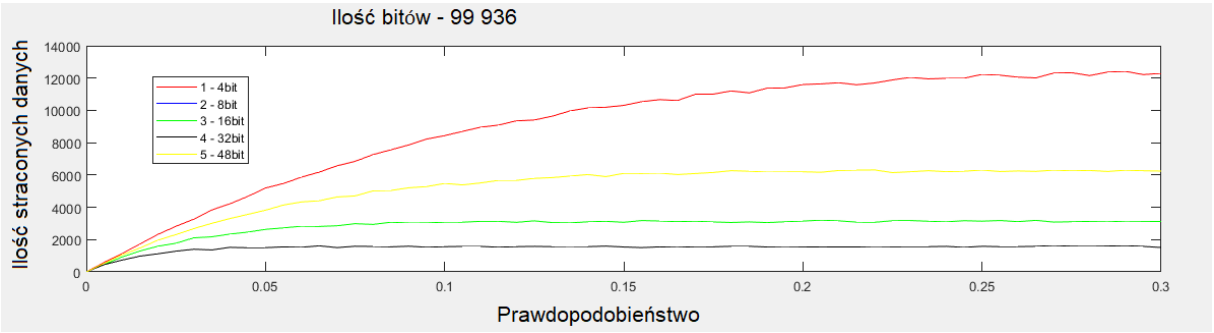
Rozdział 4

Eksperyment symulacyjny

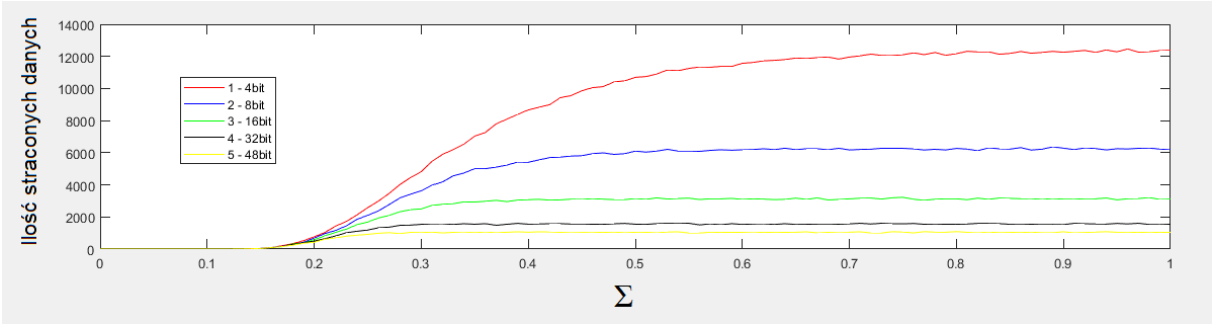
Po wstępnych założeniach, skonstruowaniu odpowiednich algorytmów i napisaniu programów symulacyjnych w programie Matlab, grupa mogła w końcu przystąpić do eksperymentu, podczas którego badała przesył bitów danych.

4.1 Bit parzystości bez kanału zwrotnego

W wersji najprostszej programu, bez kanału zwrotnego, informacja była dzielona na różne ilości pakietów. Do każdego z nich dodawany był bit parzystości, zliczający ilość jedynek w jednym pakiecie. Jeśli ilość ta była parzysta, na koniec pakietu dodawało się 1, jeśli nie, 0. W kanale prawdopodobieństwo jak i sigma były jednostajnie zmieniane. Dekoder zliczał ilość bitów przesłanych nieprawidłowo dla każdego poziomu Pp i Σ osobno. Na koniec trzeba było porównać na wykresie [6] zależność między Pp [4] (Rys. 4.1.1) lub Σ [5] (Rys.4.1.2) a ilością bitów w pakiecie.



Rys. 4.1.1 Wykres zależności ilości straconych pakietów a prawdopodobieństwem przekłamania



Rys. 4.1.2 Wykres zależności ilości straconych pakietów i Σ

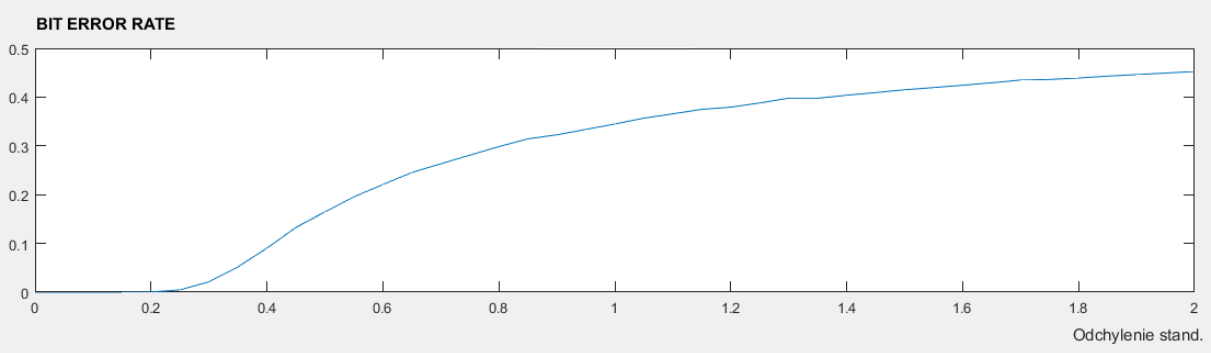
4.2 Bit parzystości z kanałem zwrotnym

Drugim zasymulowanym kanałem był już ten wymagający ponownego przesyłu danych w przypadku błędów w pakiecie. Daje to nowe możliwości ciekawych obserwacji. Można wyliczyć, ile razy dany pakiet musiał przechodzić przez kanał zwrotny z powodu zakłamań. Do tego są przydadzą się dwie nowe zmienne:

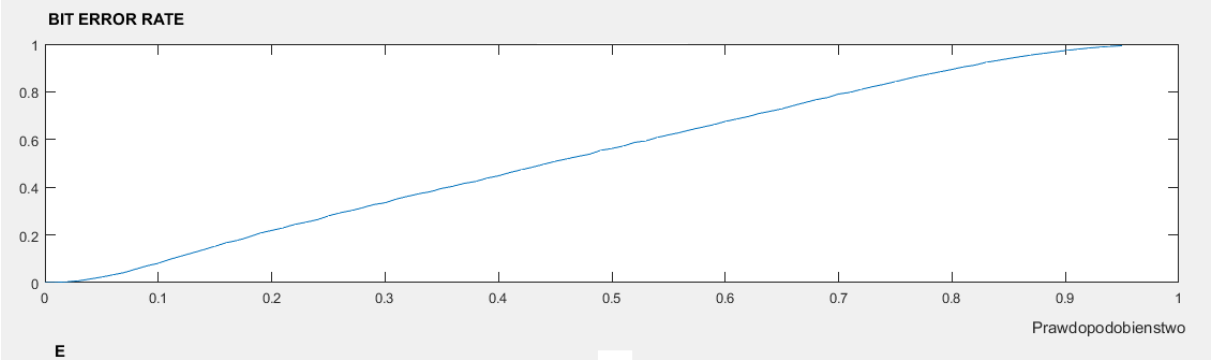
$$\mathrm{BER} = \frac{\text{liczba błędnie odebranych bitów danych}}{\text{liczba przesłanych bitów danych}}$$

$$\mathrm{E} = \frac{\text{liczba poprawnie odebranych bitów danych}}{\text{liczba przesłanych bitów danych}}$$

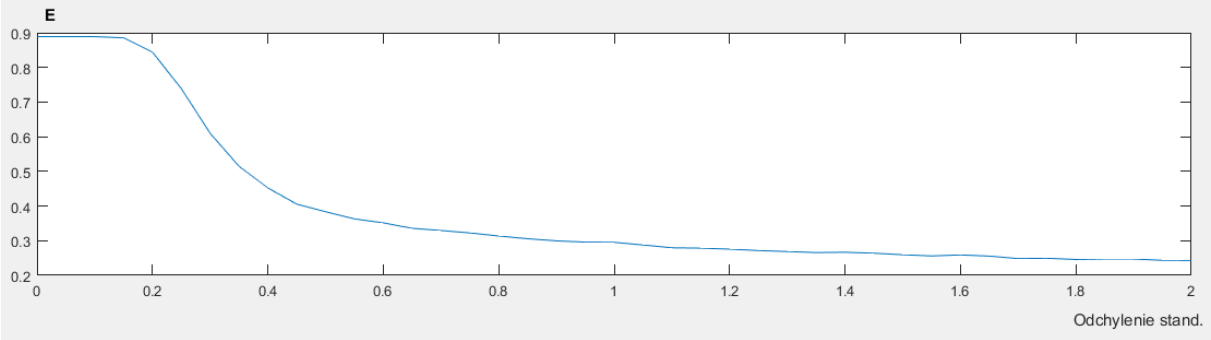
Po przesyle wszystkich bitów i obliczeniu tych zmiennych, czas na interesujące nas wykresy (Rys. 4.2.1 – 4.2.4).



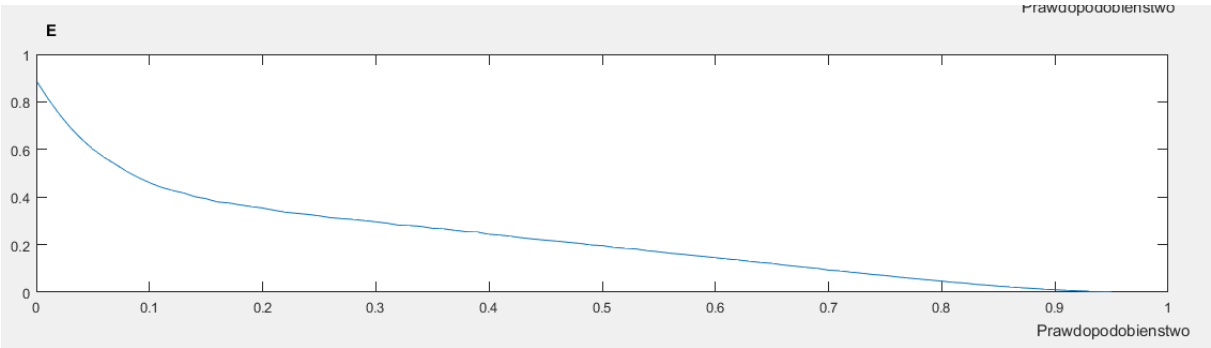
Rys. 4.2.1 Wykres zależności między BER a Σ [8]



Rys. 4.2.2 Wykres zależności między BER a prawdopodobieństwem przekłamania [7]



Rys. 4.2.3 Wykres zależności między E a Σ [8]



Rys. 4.2.4 Wykres zależności między E a prawdopodobieństwem przekłamania [7]

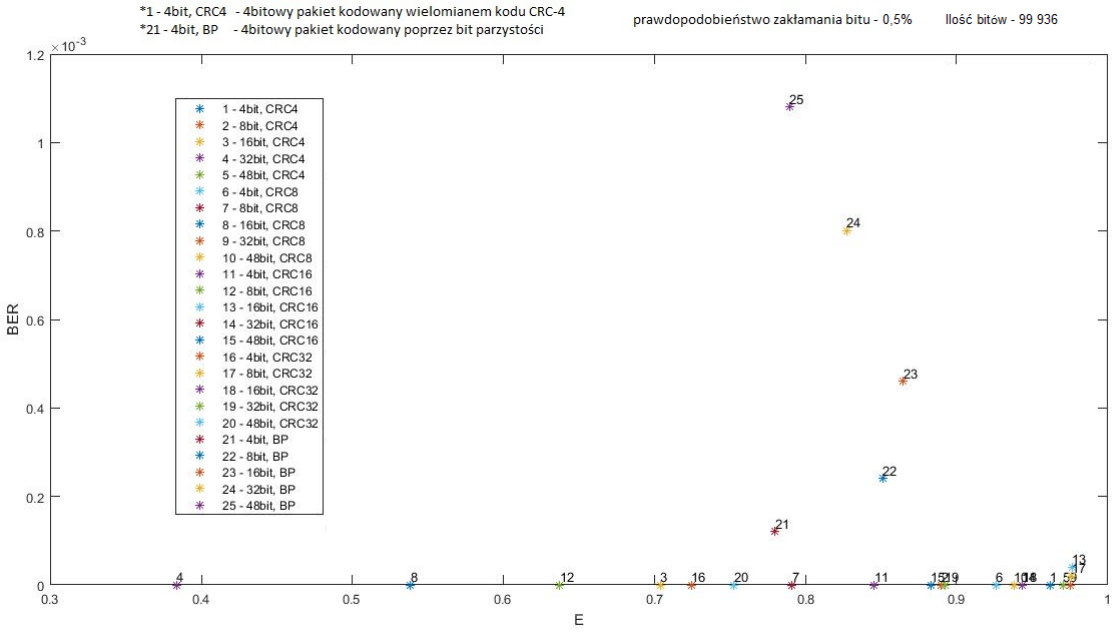
4.3 System CRC

Następnym kanałem miał być ten zawierający nadmiarowe bity dzięki wielomianowi CRC. Dla każdej informacji dzielonej na n-bitowe pakiety dany był wielomian mający n+1 składników (np. dla 5-bitowego wielomianu-dzielnika kod CRC przy wyznaczeniu reszty z dzielenia składa się z 4 bitów).

4.4 Zadanie projektowe

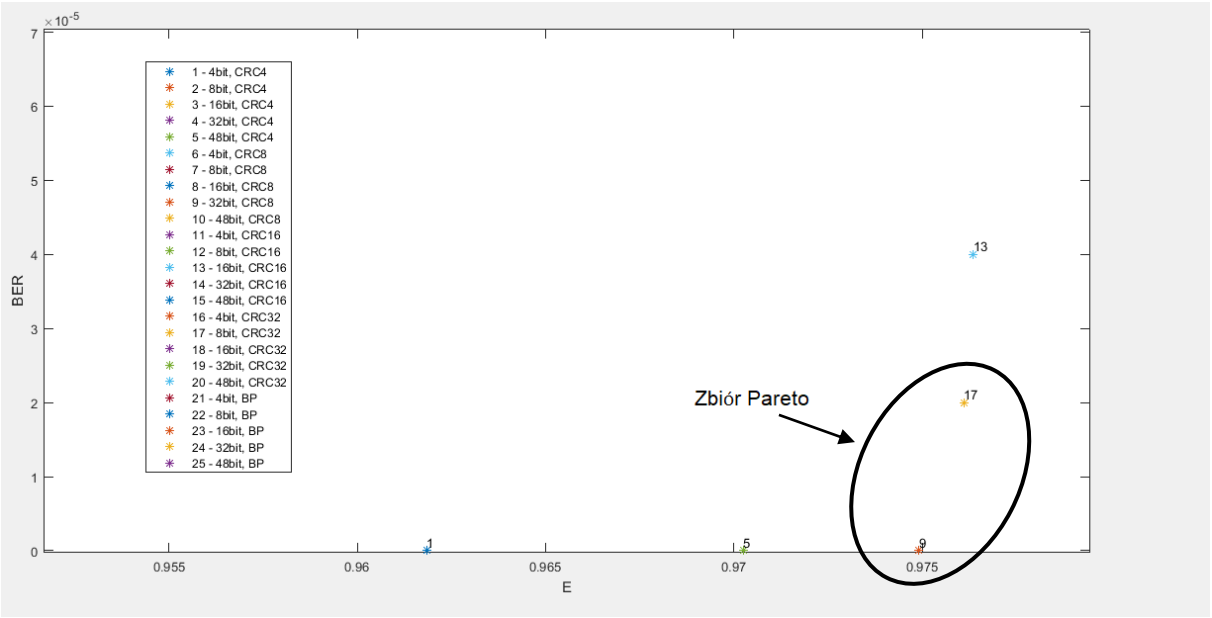
Po opracowaniu wcześniejszych algorytmów i programów nadszedł czas na użycie ich w jednym zadaniu.

Grupa, tak jak wcześniej, miała do dyspozycji system z kanałem zwrotnym. Prawdopodobieństwo przekłamania bitu tym razem było stałe i wynosiło 0,5%. Zmienną natomiast była długość pakietu i algorytm pozwalający kodować sygnał. Dla każdej wyznaczonej zmiennej obliczony został BER i E (Rys. 4.4.1).



Rys. 4.4.1 Wykres zależności między BER a E dla różnych pakietów i kodów nadmiarowych [9]

Czym mniejsza liczba błędnie odebranych pakietów i czym mniej razy dany pakiet musiał być ponownie przesyłany, tym bardziej efektywny był kod nadmiarowy. Trzeba było zatem znaleźć takie punkty, które mają jednocześnie niski współczynnik BER i wysoki współczynnik E. Najbardziej nadające się punkty zakreślone były w zbiór Pareto (Rys. 4.4.2). Oznacza to, że nie dało już się między nimi wywnioskować, który współczynnik miał większą przewagę nad drugim.



Rys. 4.4.2 Wykres z zaznaczonym zbiorem Pareto

Rozdział 5

Wnioski

Biorąc pod uwagę wyniki przedstawione na wykresach eksperymentu symulacyjnego, można dojść do kilku interesujących wniosków.

Kod nadmiarowy pod postacią bitu parzystości w porównaniu z kodem CRC jest nieefektywny. Sprawdzanie poprawności przesyłania danych tylko za pomocą jednego bitu może być niebezpieczne dla zachowania prawdziwej informacji. Przez zakłócenia w kanale pakiety mogą się zmienić, zachowując swoją parzystość. Ponadto sam bit parzystości może zostać przekłamany, niepotrzebnie sugerując niepoprawność kodu.

Symulator bez kanału zwrotnego podającego pakiety do ponownego przesłania jest symulatorem z wysokim stosunkiem Bit Error Rate. Brak możliwości poprawy danych, które wielokrotnie mogą zostać przekłamane, znacznie go podwyższa.

Podnosząc poziom prawdopodobieństwa przekłamania, wskaźnik BER wzrasta niemalże liniowo. Biorąc pod uwagę zaszumienie sygnału wykres nieco się zmienia. Gdy w kanale poziom zakłóceń jest niski (do mniej więcej $\Sigma = 0.2$), bity są nienaruszane. Nawet przy bardzo niskim Pp istnieje taka szansa, że bit się zmieni na przeciwny, w zaszumieniu praktycznie nie ma takiej sytuacji. Jest to też symulacja bliższa rzeczywistemu światu. Te same wnioski można wyciągnąć patrząc na zależność E od Σ (Rys. 4.2.3) i E od Pp (Rys. 4.2.4). Po przekroczeniu $\Sigma = 0.2$, E drastycznie spada. Dzieje się tak mniej więcej do poziomu równemu 0.6.

Analizując zachowanie kodu CRC nie można się dziwić szerokiemu zastosowaniu tego algorytmu w świecie kontroli spójności informacji. Niezależnie od przyjętej długości wielomianu, ma on bardzo niski współczynnik Bit Error Rate. Skoro różnica między wielomianami, jeśli chodzi o BER, jest często nieporównywalnie mała, pozostaje tylko kwestia ilości bitów do ponownego przesłania (czym mniej tym lepiej). Prym wiodą tu kody z krótkim wielomianem. Algorytm działa z podobną skutecznością, ale gdy kod nie zadziała, pakiet zawiera mniej danych w kanale zwrotnym, przez co transmisja jest szybsza. Zaskakującym jest więc tutaj wysoki współczynnik E kodu nr 17 (Rys. 4.4.1). Na sam koniec, używając optimum Pareto, śmiało można wyznaczyć najlepsze kody nadmiarowe, jakimi sa 8-bitowy kod CRC 32 i 32-bitowy kod CRC 8.

Bibliografia

[1] Peterson and Davie, *Computer Networks: A Systems Approach*, Third Edition, 2003

[2] [Brak autora], *Kodowanie kanałowe (nadmiarowe) Error Control Coding,* http://cygnus.tele.pw.edu.pl/potc/w12-13.pdf (dostęp 12.06.2018)

[3] McDaniel B., *An Algorithm for Error Correcting Cyclic Redundance Checks*, 01.06.2003 http://www.drdobbs.com/an-algorithm-for-error-correcting-cyclic/184401662 (dostęp12.06.2018)

Programy

[4] Bit parzystości bez kanału zwrotnego (prawdopodobieństwo)

```
function Y = niduc(m,n,pp)


%GENERATOR

iloscpakietow=m/n;
pakiety=round(rand(iloscpakietow,n));

%KODER

bity=[];
for j=1 : iloscpakietow
  jedynki=0;
    for i=1 : n
      if(pakiety(j,i)==1)
        jedynki=jedynki+1;
      end
    end
  if(mod(jedynki,2)==0)
    bity=[bity,0];
  else
    bity=[bity,1];
  end
end
bity=bity';
pakiety=[pakiety, bity];

%KANAŁ

for j=1 : iloscpakietow
  for i=1 : n+1
    if(pp>=rand(1))
      pakiety(j,i) = ~ pakiety(j,i);
    end
   end
```

```
end


%DEKODER

check=[];
for j=1 : iloscpakietow
  jedynki=0;
    for i=1 : n
      if(pakiety(j,i)==1)
        jedynki=jedynki+1;
      end
    end
  if(mod(jedynki,2)==0)
    check=[check,0];
  else
    check=[check,1];
  end
end

odebrane=pakiety(:,n+1)';
Errors=0;
for i=1 : length(bity)
  if(check(i)~=odebrane(i))
    Errors=Errors+1;
  end
end
Y=Errors;
%end
```

[5] Bit parzystości bez kanału zwrotnego (Σ)

```
function Y = niducrand(m,n,D)


%GENERATOR

iloscpakietow=m/n;
pakiety=round(rand(iloscpakietow,n));

%KODER

bity=[];
for j=1 : iloscpakietow
  jedynki=0;
    for i=1 : n
      if(pakiety(j,i)==1)
        jedynki=jedynki+1;
      end
    end
  if(mod(jedynki,2)==0)
    bity=[bity,0];
  else
    bity=[bity,1];
  end
end
bity=bity';
pakiety=[pakiety, bity];
```

```
%KANAŁ

pakiety=pakiety+D*randn(iloscpakietow,n+1);

%DEKODER

check=[];

for j=1 : iloscpakietow
  for i=1 : n+1
    if(pakiety(j,i)>=0.5)
        pakiety(j,i)=1;
    else
        pakiety(j,i)=0;
    end
  end
end

for j=1 : iloscpakietow
  jedynki=0;
    for i=1 : n
      if(pakiety(j,i)==1)
        jedynki=jedynki+1;
      end
    end
  if(mod(jedynki,2)==0)
    check=[check,0];
  else
    check=[check,1];
  end
end

odebrane=pakiety(:,n+1)';
Errors=0;

for i=1 : length(bity)
  if(check(i)~=odebrane(i))
    Errors=Errors+1;
  end
end
Y=Errors;
end
```

[6] Rysowanie wykresów do 4.1

```
subplot(2,1,1)
pp=0:0.005:0.3;
wyniki=[];
for i=pp
  wyniki=[wyniki, niduc(99936,4,i)];
end
plot(pp, wyniki,'r')
(wyniki(11))/24984
hold on;
wyniki2=[];
for i=pp
  wyniki2=[wyniki2, niduc(99936,8,i)];
```

```
end
(wyniki(11))/12492
plot(pp, wyniki2,'b')
wyniki3=[];
for i=pp
 wyniki3=[wyniki3, niduc(99936,16,i)];
end
wyniki3(11)/6246
plot(pp, wyniki3,'g')
wyniki4=[];
for i=pp
  wyniki4=[wyniki4, niduc(99936,32,i)];
end

(wyniki4(11))/3123
plot(pp, wyniki4,'k')

wyniki5=[];
for i=pp
  wyniki5=[wyniki5, niduc(99936,48,i)];
end
(wyniki(11))/2082
plot(pp, wyniki2,'y')
hold off;



subplot(2,1,2)
D=0:0.01:1;
wyniki=[];
for i=D
  wyniki=[wyniki, niducrand(99936,4,i)];
end
plot(D, wyniki,'r')
hold on;
wyniki2=[];
for i=D
  wyniki2=[wyniki2, niducrand(99936,8,i)];
end
plot(D, wyniki2,'b')
wyniki3=[];
for i=D
 wyniki3=[wyniki3, niducrand(99936,16,i)];
end
plot(D, wyniki3,'g')
wyniki4=[];
for i=D
  wyniki4=[wyniki4, niducrand(99936,32,i)];
end
plot(D, wyniki4,'k')
wyniki5=[];
for i=D
  wyniki5=[wyniki5, niducrand(99936,48,i)];
end
plot(D, wyniki5,'y')
hold off;
```

[7] Zależność między BER i E a prawdopodobieństwem przekłamania

```
BER=[];
e=[];
h=[];
n=8;
m=99936;
for pp=0:0.01:0.95
odebrane=[];
ciagbitow=gen(m,n);
kod=koder(ciagbitow);
E=0;
Errors=0;
k=1;
while (k<=m/n)
    kan=kanal(kod,pp,k);
    dek=dekoder(kan);
    if(dek==0)
      k=k+1;
      odebrane=[odebrane; kan];
    else
      E = E + 1;
    end

end

[a,b]=size(odebrane);

for d=1:a
  for j=1: b
    if(odebrane(d,j)~=kod(d,j))
    Errors = Errors + 1;
    end
  end
end

h=[h,Errors];
BER=[BER, Errors/m];
e=[e,(m-Errors)/(m+m/n+E*(n+1))];

end

pp=0:0.01:0.95;
subplot(2,1,1)
plot(pp,BER)
title('BIT ERROR RATE')
xlabel('Prawdopodobienstwo')
subplot(2,1,2)
plot(pp,e)
title('E')
xlabel('Prawdopodobienstwo')
```

[8] Wykres zależności między BER i E a odchyleniem standardowym

```
BER=[];
e=[];
h=[];
n=8;
m=99936;
for pp=0:0.05:2
odebrane=[];
ciagbitow=gen(m,n);
kod=koder(ciagbitow);
E=0;
Errors=0;
k=1;
while (k<=m/n)
    kan=kanalrand(kod,pp,k);
    dek1=dekoderand(kan);
    dek=dek2(dek1);
    if(dek==0)
      k=k+1;
      odebrane=[odebrane; dek1];
    else
      E = E + 1;
    end

end

[a,b]=size(odebrane);

for d=1:a
  for j=1: b
    if(odebrane(d,j)~=kod(d,j))
    Errors = Errors + 1;
    end
  end
end

h=[h,Errors];
BER=[BER, Errors/m];
e=[e,(m-Errors)/(m+m/n+E*(n+1))];

end

pp=0:0.05:2;
subplot(2,1,1)
plot(pp,BER)
title('BIT ERROR RATE')
xlabel('Odchylenie stand.')
subplot(2,1,2)
plot(pp,e)
title('E')
xlabel('Odchylenie stand.')
```

[9] Program i wykresy zależności BER i E dla kodów CRC i kodów BP

```
      numb = 0;

for n = [4 8 16 32 48]
    for crck = 1:1:4



if(crck == 1)
    poly = ([1 0 0 1 1]);  %wielomian do CRC
elseif(crck == 2)
    poly = ([1 1 1 0 1 0 1 0 1]);
elseif(crck == 3)
    poly = ([1 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 1]);
elseif(crck == 4)
    poly = ([1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 1 0 1 1 0 1 1 0 1
1 1]);
end



m=99936;   %ilosc bitow



z = 1;
zet = 1;
E = 0;  %ilosc powtornie przeslanych pakietow przez wzglad bledow
BER = 0; %BitErrrorRate
pp = 0.005;  %prawdopodobienstwo wystapienia bledu
Errors = 0;  %ilosc bledow
gen = crc.generator(poly);  %wyznaczenie wielomianu dla kodowania
det = crc.detector(poly);   %wyznaczenie wielomianu dla dekodowania
iloscpakietow = m/n;
pakiety = round(rand(iloscpakietow,n));
ciagbitow=pakiety';
encoding = zeros(n+(length(poly)-1),(m/n));
odebrane = zeros(n+(length(poly)-1),(m/n));

b = n+1:1:n+(length(poly)-1);

while (z <(m/n))                                              %koder
    encoding(:,z) = generate(gen, ciagbitow(:,z));
    z = z+1;
end                                                           %%

truecode = encoding;


while (zet <(m/n))                                            %kanal
    for er = 1 : 1 : n + (length(poly)-1)
         if(pp>=rand(1))
            encoding(er,zet) = ~truecode(er,zet);
         end
```

```
    end                                                     %%

    [outdata error] = detect(det, encoding(:,zet));  %dekoder, sprawdzanie
bledow
        if (error == 0)
            odebrane(:,zet) = encoding(:,zet);
            zet = zet+1;
        else
            E = E+1;
            encoding(:,zet) = truecode(:,zet);
        end                                                 %%
end




for p = 1 : 1 : (m/n)
        for r = 1 : 1 : n
            if(odebrane(r,p) ~= truecode(r,p))
                Errors = Errors + 1;
            end
        end
end

BER = Errors/m;
e = ((m-Errors)/(m+(n+(length(poly)-1)*E)));
plot(e,BER,'*')
numb = numb + 1;
numb = num2str(numb);
text(e,BER,numb,'VerticalAlignment','bottom','HorizontalAlignment','left');
numb = str2num(numb);
numb
hold on

    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for n = [4 8 16 32 48]
    BER=[];
e=[];
h=[];
m=99936;
pp=0.005;
odebrane=[];
iloscpakietow=m/n;
pakiety=round(rand(iloscpakietow,n));
ciagbitow = pakiety;
kod=koder(ciagbitow);
E=0;
Errors=0;
k=1;
while(k<m/n)
    kan=kanal(kod,pp,k);
    dek=dekoder(kan);
    if(dek==0)
      k=k+1;
      odebrane=[odebrane; kan];
    else
      E = E + 1;
```

```
    end

end

[a,b]=size(odebrane);

for d=1:a
  for j=1: b
    if(odebrane(d,j)~=kod(d,j))
    Errors = Errors + 1;
    end
  end
end

h=[h,Errors];
BER=[BER, Errors/m];
e=[e,(m-Errors)/(m+m/n+E*(n+1))];

plot(e,BER,'*')

numb = numb + 1;
numb = num2str(numb);
text(e,BER,numb,'VerticalAlignment','bottom','HorizontalAlignment','left');
numb = str2num(numb);
numb
hold on
xlabel('E');
ylabel('BER');
if n == 48
    legend('1 - 4bit, CRC4', '2 - 8bit, CRC4', '3 - 16bit, CRC4', '4 -
32bit, CRC4', '5 - 48bit, CRC4', '6 - 4bit, CRC8', '7 - 8bit, CRC8', '8 -
16bit, CRC8', '9 - 32bit, CRC8', '10 - 48bit, CRC8', '11 - 4bit, CRC16',
'12 - 8bit, CRC16', '13 - 16bit, CRC16', '14 - 32bit, CRC16', '15 - 48bit,
CRC16', '16 - 4bit, CRC32', '17 - 8bit, CRC32', '18 - 16bit, CRC32', '19 -
32bit, CRC32', '20 - 48bit, CRC32', '21 - 4bit, BP', '22 - 8bit, BP','23 -
16bit, BP', '24 - 32bit, BP', '25 - 48bit, BP')
end
end
```