NIEZAWODNOŚĆ I DIAGNOSTYKA UKŁADÓW CYFROWYCH 2 – PROJEKT SPRAWOZDANIE

temat projektu:

SCRAMBLING

Prowadzący:

dr. inż. Marek Woda

Termin zajęć projektowych:

piątek TN 10:15 – 13:00

Grupa projektowa:
Sebastian Jantos 225982
Maksymilian Maczurek 225976
Bartosz Pogoda 225988

Spis treści

1. Wstęp

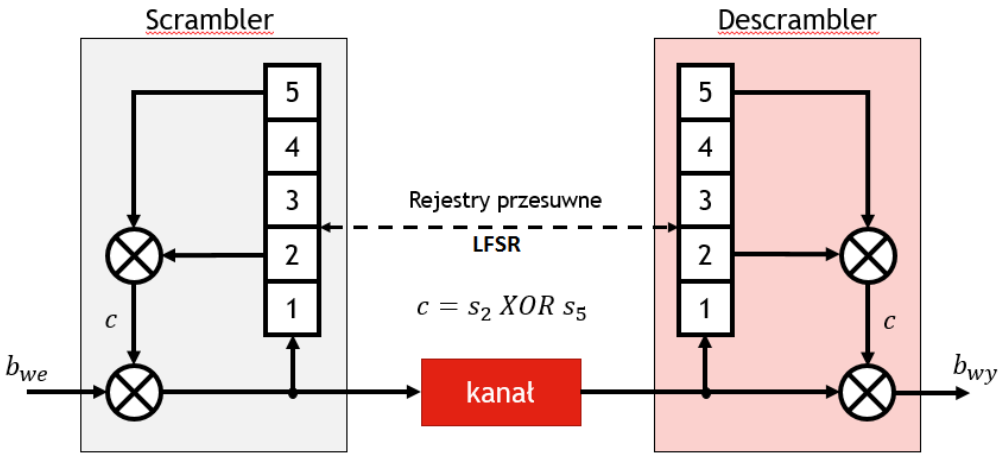
Otrzymane zadanie projektowe polegało na stworzeniu symulatora scramblera. Podczas transmisji danych najważniejsze jest to, aby dane w odbiorniku były jak najbardziej zbliżone do tych wysyłanych przez nadajnik. Związane jest to z synchronizacją danych (bitów) – gdzie istotne jest rozwiązanie problemu pojawiających się w sygnale długich ciągów zer lub jedynek. Rozwiązaniem jest zastosowanie scramblera, dzięki któremu część bitów zostaje zmieniona w stosunku do ciągu wejściowego.

Scrambler jest to urządzenie, którego zadaniem jest randomizowanie dostarczonego sygnału. Oznacza to, że sygnał, który otrzymuje się na wyjściu jest zmieniony w pewien losowy sposób. Operacja ta służy usunięciu z sygnału długich ciągów zer i jedynek. Jest to istotne, ponieważ odbiornik podczas długiej sekwencji tego samego stanu nie daje gwarancji, że odebrany ciąg będzie tej samej długości co w sygnale nadanym. W najgorszym przypadku przy długiej sekwencji zer odbiornik mógłby przestać odbierać dane odczytując ciągły stan niski jako brak transmisji. Zjawisko błędnego odbierania bitów nadawanych przez nadajnik nazywane jest *desynchronizacją*. Aby zapobiec takim błędom w transmisji wprowadza się do układu scramblery, które zamieniają sygnał w taki sposób, aby uniknąć ciągów tych samych stanów. Dzięki częstszym zmianom stanów, szansa na błędny odbiór danych spada, ponieważ z każdą zmianą stanu, zegar taktujący w odbiorniku się odświeża.

Poza błędami wynikającymi z desynchronizacji nadajnika i odbiornika, w rzeczywistości pojawią się także przekłamania w trakcie przesyłania sygnału. Mogą one mieć różne źródła, zakłócenia elektromagnetyczne, zjawiska atmosferyczne itp. Bez względu na ilość i rodzaj źródeł zakłóceń, szansę na pojawienie się błędu można określić za pomocą jednej liczby wyrażającej prawdopodobieństwo zakłamania. Ten rodzaj zakłóceń będzie modelowany za pomocą Binary Symmetric Channel. Jednak podczas transmisji mogą się także pojawić specyficzne rodzaje błędów np. periodyczne zakłamania, zakłamania kilku pierwszych lub ostatnich bitów. Mogą wynikać one choćby z awarii urządzeń. Wszystkie wymienione tu rodzaje błędów mogą zostać przetestowane w symulatorze stworzonym w ramach tego projektu.

Przesyłane dane muszą także podlegać jakimś normom kodowania, w przypadku tego projektu użyte zostało kodowanie 64b/66b (kodowanie Ethernet). Oznacza to, że sygnał podzielony jest na segmenty po 64 bity, a na początek każdego z nich dodawane są dwa bity preambuły. Preambuła musi mieć postać 01 po których jest 64 bitów danych, lub 10, a następnie 8 bitów nadmiarowych i 56 bitów danych. Jeśli zatem na spodziewanej pozycji (wielokrotności liby 65 i 66) nie występuje prawidłowa preambuła, to nastąpiła desynchronizacja, którą trzeba w miarę możliwości naprawić.

Kiedy sygnał zostaje odebrany, nie jest to oryginalny sygnał, tylko losowo zmieniony a do tego możliwe, że w pewnych miejscach zakłamany. Błędy powstałe w wyniku zakłóceń można próbować skorygować za pomocą pewnych algorytmów, jednak wciąż, aby odczytać sygnał, należy cofnąć zmiany wprowadzone przez scrambler. Jest to zadanie descramblera, który tak naprawdę jest takim samym urządzeniem co scrambler. Urządzenia scramblujące podzielić można na addytywne, które wymagają dodatkowej synchronizacji oraz multiplikatywne, czyli samo synchronizujące się. W tym projekcie wykonano symulację scramblera multiplikatywnego, ponieważ taki jest używany w kodowaniu ethernetowym 64b/66b. W jego skład wchodzi kilka bramek logicznych XOR oraz LFSR, czyli Linear Feedback Shift Register (liniowy zwrotny rejestr przesuwny). Schemat pokazany na [Rysunek 1] obrazuje budowę tego urządzenia.



$$((x\ XOR\ y)\ XOR\ y) = x$$

Rysunek 1. Schemat budowy scramblera oraz descramblera multiplikatywnego

Każdy bit przesyłanego sygnału jest zmieniany za pomocą pewnej kombinacji bramek XOR operujących na wartościach w rejestrze. Następnie w przypadku scramblera multiplikatywnego, wyliczona wartość bitu jest ładowana do rejestru i przesyłana. Po stronie descramblera otrzymany bit najpierw jest przywracany do stanu oryginalnego za pomocą tej samej kombinacji bramek, a dopiero potem odebrany, zrandomizowany bit jest ładowany do rejestru. Ponieważ preambuły segmentów nie mogą być zmienione, oznacza to, że sygnał jest scramblowany przed zakodowaniem. W scramblerze oraz descramblerze, aby mogły działać, należy ich rejestry zainicjować takimi samymi wartościami. W przypadku scramblerów addytywnych, wymagających synchronizacji należy te rejestry przywracać do początkowego stanu co każdą preambułę, jednak w przypadku scramblera samo synchronizującego się wystarczy to zrobić raz na początku transmisji. Oszczędza to pewnych trudności z przeładowywaniem rejestrów, jednak takie podejście powoduje powielanie się błędów w odczytywanym sygnale.

W wybranej technologii (Ethernet) w scramblerze używa się funkcji $G(x) = x^{58} + x^{39} + 1$ co oznacza, że bramki XOR czerpią dane z 58, 39 i 0 pozycji rejestru LFSR.

W celu zaprojektowania i symulacji układu scramblera użyto środowiska ***Matlab R2017a***, ponieważ program poprzez swój szeroki zasób narzędzi pozwala na przeprowadzenie skomplikowanych obliczeń matematycznych. Do przeprowadzenia analizy danych oraz stworzenia wykresów użyto programu z pakietu ***MS Office Excel 2016.*** Na potrzeby projektu użyto jedynie część dostępnych narzędzi w wyżej wymienionych środowiskach.

1.1. Zadania projektowe

- Implementacja symulatora pozwalającego na modelowanie desynchronizacji oraz dodawanie przekłamań bitów za pomocą Binary Symetric Channel (BSC).
- Implementacja prostego scramblera oraz użycie kodowania 64b/66b (Ethernet).
- Stworzenie mechanizmu pozwalającego na odzyskanie synchronizacji w przypadku wykrycia jej utracenia.

(źródło: http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/ndsc/mat_proj/mat_proj_zadania_ps/)

1.2. Słowniczek pojęć

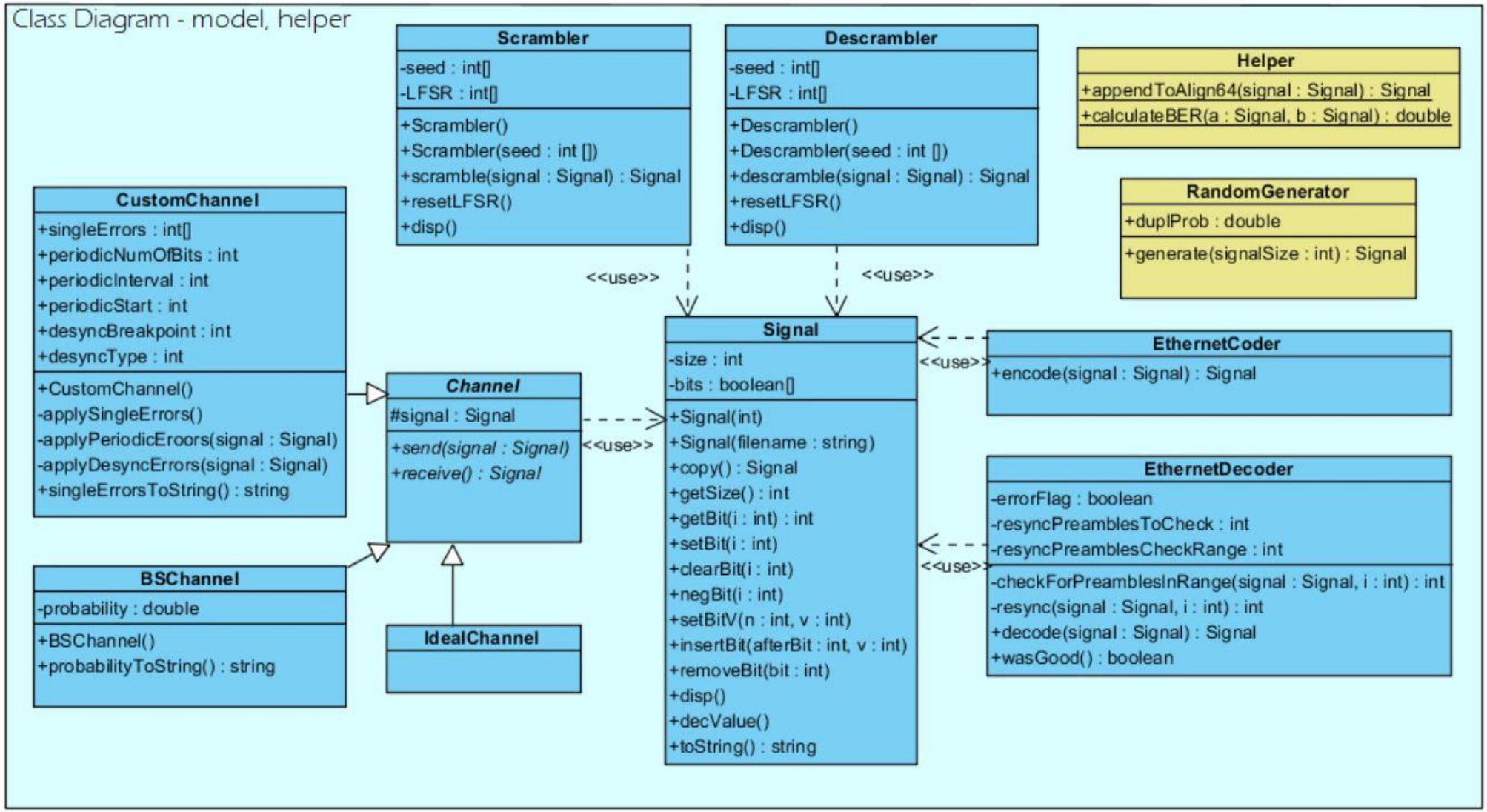
BER - (ang. *Bit Error Rate*) – współczynnik ilości otrzymanych błędnych bitów do całkowitej liczby otrzymanych bitów

LFSR (ang. Linear Feedback Shift Register) – rejestr przesuwający, którego wejściowy bit jest funkcją liniową jego poprzedniego stanu.

1.3. Usługi koordynujące prace przy projekcie

- GitHub – w celu kontroli wersji i koordynacji pracy nad tworzonym kodem źródłowym symulatora
- Trello – tablica w celu zapisu wszystkich pomysłów, zebranych pomocy naukowych, zapis planu działania

2. Diagram klas



3. Opis użytych algorytmów w projekcie

Poniżej zostały opisane najważniejsze algorytmy zaimplementowane przy tworzeniu symulatora.

3.1. Algorytm kodowania oraz dekodowania Ethernet

Kodowanie 64b/66b

Pierwsze dwa bity służą do synchronizacji i mogą przyjmować wartość:

- 01, gdy przesyłane są tylko dane, reszta danych jest zascramblowana
- 10, następnie 8 bitów określa typ i pozostałe 56 bitów jest zescramblowane.

W ramach projektu zostało przyjęte uproszczenie polegające na używaniu jedynie preambuły ‘01’ oraz 64 bitów danych w ramce. Dla takiego założenia użyty dekoder Ethernet dla napotkanej preambuły ‘10’ zgłasza błąd synchronizacji oraz rozpoczyna resynchronizację.

Pseudokod algorytmu kodowania (signal – sygnał do zakodowania):

```
   i := 0                                              - iterator po sygnale

while i+1 <= signal.getSize() do
  signal.insertBit(i, 0)                               - wstawienie preambuły 01
  signal.insertBit(i+1, 1)                        -

  i := i + 66                                          - pominięcie wstawionej preambuły
                                                       - oraz 64 bitów danych
end
```

Opis słowny algorytmu dekodowania:

```
1. Zresetowanie flagi desynchronizacji

2. Ustawienie iteratora na początku dekodowanego sygnału

3. Póki w zasięgu sygnału:

   2.1. Sprawdzenie poprawności preambuły

   2.2. Jeśli poprawna:

     2.2.1. Skopiowanie następujących po niej 64 bitów danych
            do wyjściowego sygnału

   2.3. Jeśli niepoprawna:

     2.3.1. Ustawienie flagi desynchronizacji

     2.3.2. Wykonanie algorytmu resynchronizacji

     2.3.3. Skopiowanie 64 bitów danych rozpoczynając od indeksu
            zwróconego przez algorytm resynchronizacji.
   2.4. Przesunięcie iteratora o 66 bitów do przodu.
```

3.2. Algorytm desynchronizacji

Desynchronizacja w rzeczywistych kanałach występuje, gdy następuje duża ilość powtórzeń jednego symbolu pod rząd – odbiornik przez dłuższy czas nie odnotowuje żadnych zmian w odbieranym sygnale, co może prowadzić do błędów w interpretacji takiego sygnału. Desynchronizacja została zamodelowana oraz zaimplementowana na potrzeby badań w konfigurowalnym kanale (CustomChannel).

Parametry:

- desyncBreakpoint – minimalna ilość takich samych bitów występujących pod rząd, która spowoduje wystąpienie desynchronizacji.
- desyncType – typ desynchronizacji, czyli dodanie powtarzającego się bitu lub jego usunięcie.

Dane wejściowe, wyjściowe:

- signal – sygnał przesyłany przez kanał

Pseudokod algorytmu:

```
repeatState := -1                                   - aktualnie powtarzany symbol
repeatCounter := 0                                  - licznik powtórzeń
i := 0

while i <= signal.getSize() do
  if signal.getBit(i) == repeatState then           -  jeżeli napotkany symbol jest
    repeatCounter := repeatCounter + 1              -  aktualnie zliczanym, zwiększenie
  else                                              -  licznika, w przeciwnym razie
    repeatCounter := 0                              -  wyzerowanie licznika.
  endif

  if repeatCounter == 0 then                        -  licznik == zero oznacza, że nastąpiła
    repeatState := signal.getBit(i)                 -  zmiana zliczanego symbolu
    repeatCounter := 1                              -  uwzględniamy nowo zliczany symbol
  endif

  if repeatCounter == desyncBreakpoint then
    if desyncType == 1 then                         -  gdy zliczymy zadaną ilość powtórzeń
      signal.insertBit(i, repeatState)              -  symbolu, w zależności od typu
      i := i+1                                      -  desynchronizacji, wstawiamy bądź
    elseif desyncType == -1                         -  usuwamy jeden symbol
      signal.removeBit(i)                           -  oraz odpowiednio modyfikujemy indeks
      i := i-1                                      -  sygnału uwzględniając uzyskane
    elseif desyncType == 2                          -  przesunięcie
      if round(rand()) then
        signal.insertBit(i, repeatState)            -  desyncType == -1    usunięcie symbolu
        i := i+1                                    -  desyncType == 1      dodanie symbolu
      else                                          -  desyncType == 2   losowe dodanie lub
        signal.removeBit(i)                         -                     usunięcie symbolu
        i := i-1
      endif
    endif
    repeatCounter := 0                              -  wyzerowanie licznika powtórzeń.
  endif
  i := i + 1                                        -  przejście na następny bit sygnału
endwhile
```

3.3. Algorytm resynchronizacji

Użycie sprawnego algorytmu odzyskiwania synchronizacji jest kluczowe, aby czerpać korzyści ze stosowania kodowania Ethernet 64b/66b. O ile wykrycie nastąpienia desynchronizacji jest trywialne – wystarczy sprawdzić czy w oczekiwanym miejscu znalazła się poprawna preambuła, o tyle odnalezienie prawdziwej, przesuniętej preambuły w celu ponownej synchronizacji kanału okazuje się zadaniem bardzo trudnym. Problem jest na tyle skomplikowany, że algorytmy rozwiązujące go okazały się być trudno dostępne – przeznaczone do użytku komercyjnego.

Na potrzeby badań został zaprojektowany oraz zaimplementowany „naiwny" algorytm. (EthernetDecoder)

Parametry:

- resyncPreamblesToCheck – ilość „potencjalnych" preambuł do sprawdzenia (w obydwie strony)
- resyncPreambleCheckRange – zasięg szukania pasujących preambuł

Dane wejściowe:

- signal – dekodowany sygnał
- badPreambleIndex – indeks bitu rozpoczynającego wykrytą „złą" preambułę

Dane wyjściowe:

- dataIndex – wyznaczony pierwszy bit danych – następujący po znalezionej preambule

Opis słowny algorytmu:

1. Utwórz tablicę indeksów znalezionych preambuł, wypełnij -1.
2. Utwórz tablicę ocen znalezionych preambuł, wypełnij zerami.
3. Poszukiwanie preambuł na poprzednich pozycjach („w lewo")
   3.1. Dla każdej znalezionej preambuły:
      3.1.1. Sprawdzaj we wszystkich obszarach odległych o 66 zarówno poprzedzających jak i następujących czy w zasięgu +- resyncPreambleCheckRange występuje prawidłowa preambuła. Jeśli tak – dodaj do sumarycznej oceny preambuły ocenę wyliczoną na podstawie odległości preambuły od środka zakresu – im bliżej tym większy wynik. Brak preambuły w zakresie daje ujemny wynik.
4. Analogicznie poszukiwanie preambuł na następnych pozycjach („w prawo")
5. Wybranie preambuły o najlepszej sumarycznej ocenie.
6. Pierwszy bit danych (dataIndex) = indeks najlepszej preambuły + 2

3.4. Algorytm scramblera i descramblera

W rzeczywistości scrambler i descrambler są częściami hardware'u, czyli są realnymi urządzeniami, a nie oprogramowaniem. Nie są skomplikowane, a dzięki takiej realizacji można otrzymać bardzo wysoką wydajność przetwarzania sygnału. Symulator jednak jest tylko oprogramowaniem, więc symulowane urządzenie będzie operować na sygnale reprezentowanym przez ciąg wartości logicznych. Za pomocą kilku prostych logicznych funkcji i tablicy na rejestr udało się w środowisku MATLAB zasymulować działanie scramblera i descramblera.

Poniżej pokazano algorytm scramblera i descramblera, jest między nimi drobna różnica wynikająca z faktu, że funkcja licząca na bramkach XOR w obu przypadkach musi liczyć na takim samym stanie rejestru LFSR, stąd różnica w kolejności ładowania bitu do rejestru.

Parametry:

- LFSR – początkowa wartość, którą jest inicjalizowany rejestr

Dane wejściowe:

- inputSignal – scramblowany sygnał

Dane wyjściowe:

- outputSignal – zrandomizowany sygnał

Opis słowny algorytmu scramblowania:

```
1. Stwórz rejestr (tablicę) i wpisz do niej LFSR.

2. Dla każdego bitu w inputSignal

      2.1. bit = XOR(LFSR[0], XOR(LFSR[39], LFSR[58]))
      2.2. bitSygnału = XOR(bitSygnału, bit)
      2.3. Przesuń LFSR (0->1->2->3->…->57->58)
      2.4. LFSR[0] = bitSygnału.
      2.5. Dodaj obliczony bitSygnału do outputSignal

3. Zwróć outputSignal
```

Opis słowny algorytmu descramblowania i jego różnica (punkt 2.0 i 2.4)

```
1. Stwórz rejestr (tablicę) i wpisz do niej LFSR.

2. Dla każdego bitu w inputSignal

      2.0. Zapamiętaj bit odebrany w insertBit
      2.1. bit = XOR(LFSR[0], XOR(LFSR[39], LFSR[58]))
      2.2. bitSygnału = XOR(bitSygnału, bit)
      2.3. Przesuń LFSR (0->1->2->3->…->57->58)
      2.4. LFSR[0] = insertBit.
      2.5. Dodaj obliczony bitSygnału do outputSignal

3. Zwróć outputSignal
```

3.5. Symulator – testy

Testy symulatora polegały na przeprowadzeniu testów dla różnych kanałów i porównaniu współczynników BER w zależności od zastosowanych przekształceń sygnału:

1. Transmisja poprzez kanał, w którym występuje desynchronizacja (losowo dodanie lub usunięcie symbolu przy 12 powtórzeniach). Zbadanie zależności BER od parametrów przesyłanego sygnału – prawdopodobieństwa powtórzeń bitów. [Wykres 1.] [Wykres2.]

2. Transmisja poprzez kanał BSC. Zbadanie zależności BER od prawdopodobieństwa przekłamania bitu w kanale. [Wykres 3.] [Wykres 4.]

3. Transmisja poprzez kanał mieszany (dwa powyższe kanały). Zbadanie zależności BER od parametrów przesyłanego sygnału oraz prawdopodobieństwa przekłamań bitów w kanale BSC. [Wykres 5.] [Wykres 6.]

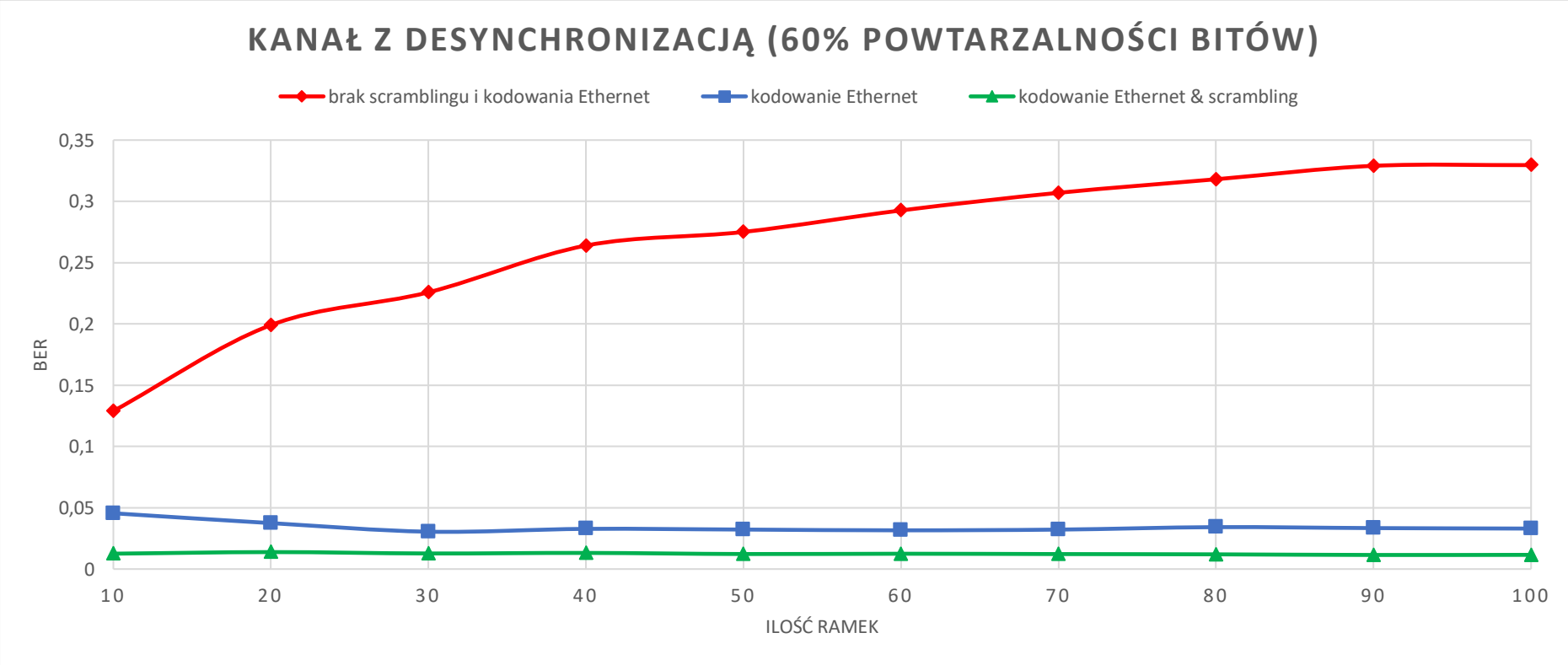
Każdy wylosowany sygnał został przesłany przez kanał:

1. Bez żadnych przekształceń (na wykresach oznaczono kolorem czerwonym)
2. Z użyciem kodowania Ethernet (na wykresach oznaczono kolorem niebieskim)
3. Z użyciem scramblingu oraz kodowania Ethernet (na wykresach oznaczono kolorem zielonym)

W każdym przypadku **wykonano 1000 iteracji** każdego testu, a następnie **wyciągnięto średnią wartość BER**. W dalszej części sprawozdania przedstawiono uzyskane wyniki w formie tabeli oraz wykresu prezentującego dany test.

3.5.1. Testy – kanał z desynchronizacją i danymi wejściowymi z tendencją do powtórzeń bitów

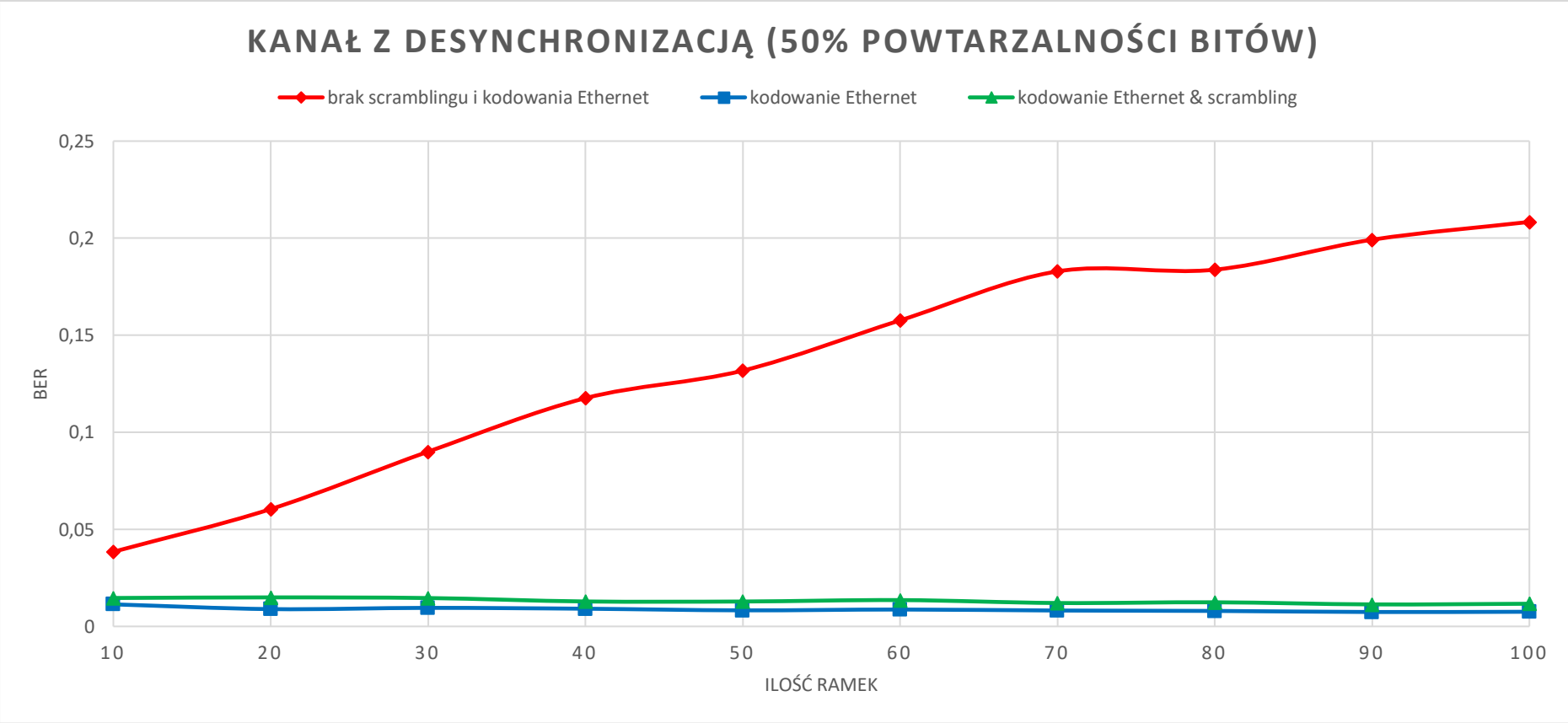
ILOŚĆ RAMEK 64b	10 (640b)	20 (1280b)	30 (1920b)	40 (2460b)	50 (3200b)	60 (3840b)	70 (4480b)	80 (5120b)	90 (5760b)	100 (6400b)
BRAK SCRAMBLINGU [BER]	0,129	0,200	0,226	0,264	0,276	0,293	0,307	0,319	0,329	0,330
ETHERNET [BER]	0,046	0,038	0,031	0,033	0,033	0,032	0,033	0,035	0,034	0,034
SCRAMBLING & ETHERNET [BER]	0,013	0,014	0,013	0,014	0,013	0,013	0,013	0,013	0,012	0,012



Wykres 1. Testy – kanał z desynchronizacją i danymi wejściowymi z tendencją do powtórzeń bitów

3.5.2. Testy – kanał z desynchronizacją i zrandomizowanymi danymi wejściowymi

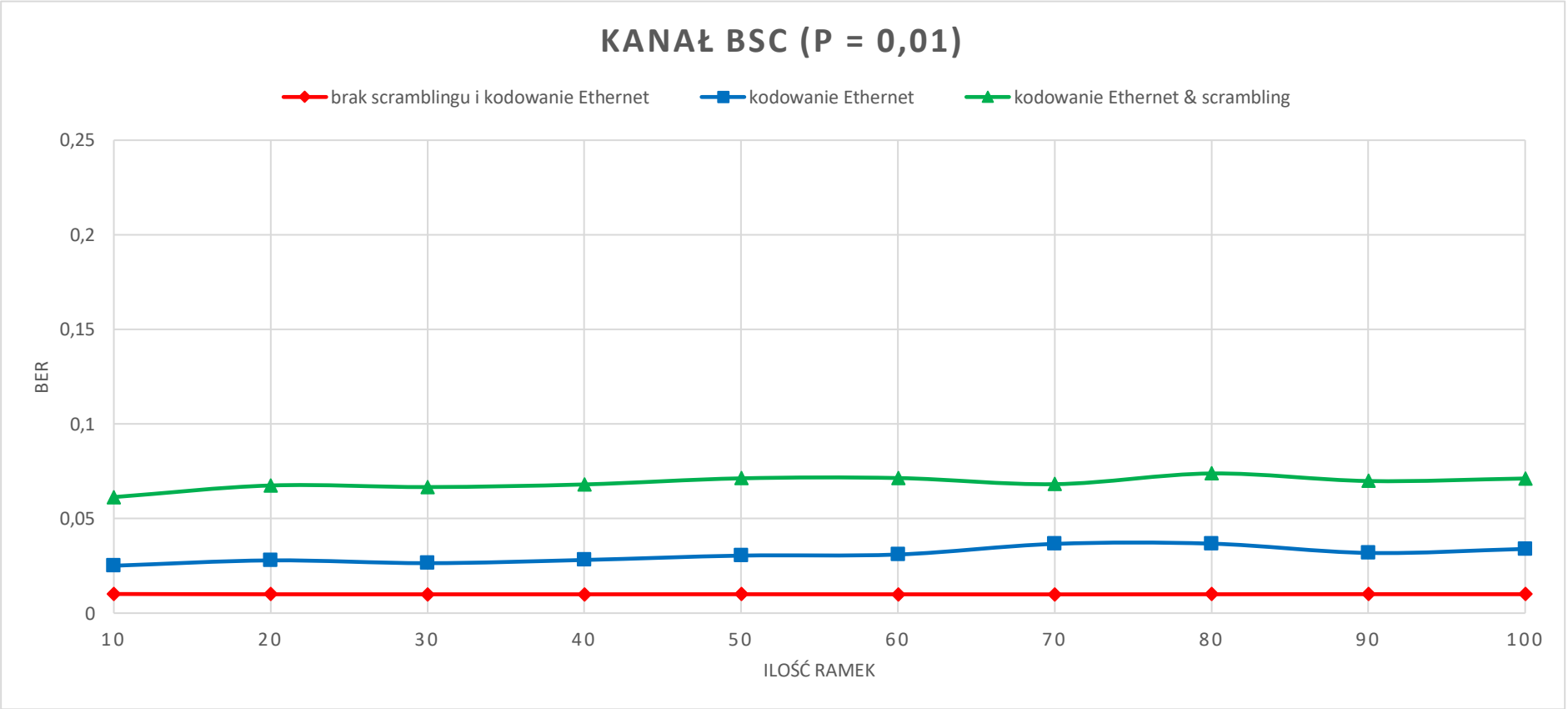
ILOŚĆ RAMEK 64b	10 (640b)	20 (1280b)	30 (1920b)	40 (2460b)	50 (3200b)	60 (3840b)	70 (4480b)	80 (5120b)	90 (5760b)	100 (6400b)
BRAK SCRAMBLINGU [BER]	0,039	0,061	0,091	0,118	0,132	0,158	0,183	0,184	0,200	0,209
ETHERNET [BER]	0,012	0,009	0,010	0,010	0,009	0,009	0,009	0,008	0,008	0,008
SCRAMBLING & ETHERNET [BER]	0,015	0,015	0,015	0,013	0,013	0,014	0,013	0,013	0,012	0,012



Wykres 2. Testy – kanał z desynchronizacją i zrandomizowanymi danymi wejściowymi

3.5.3. Testy – kanał BSC z parametrem p = 0,01

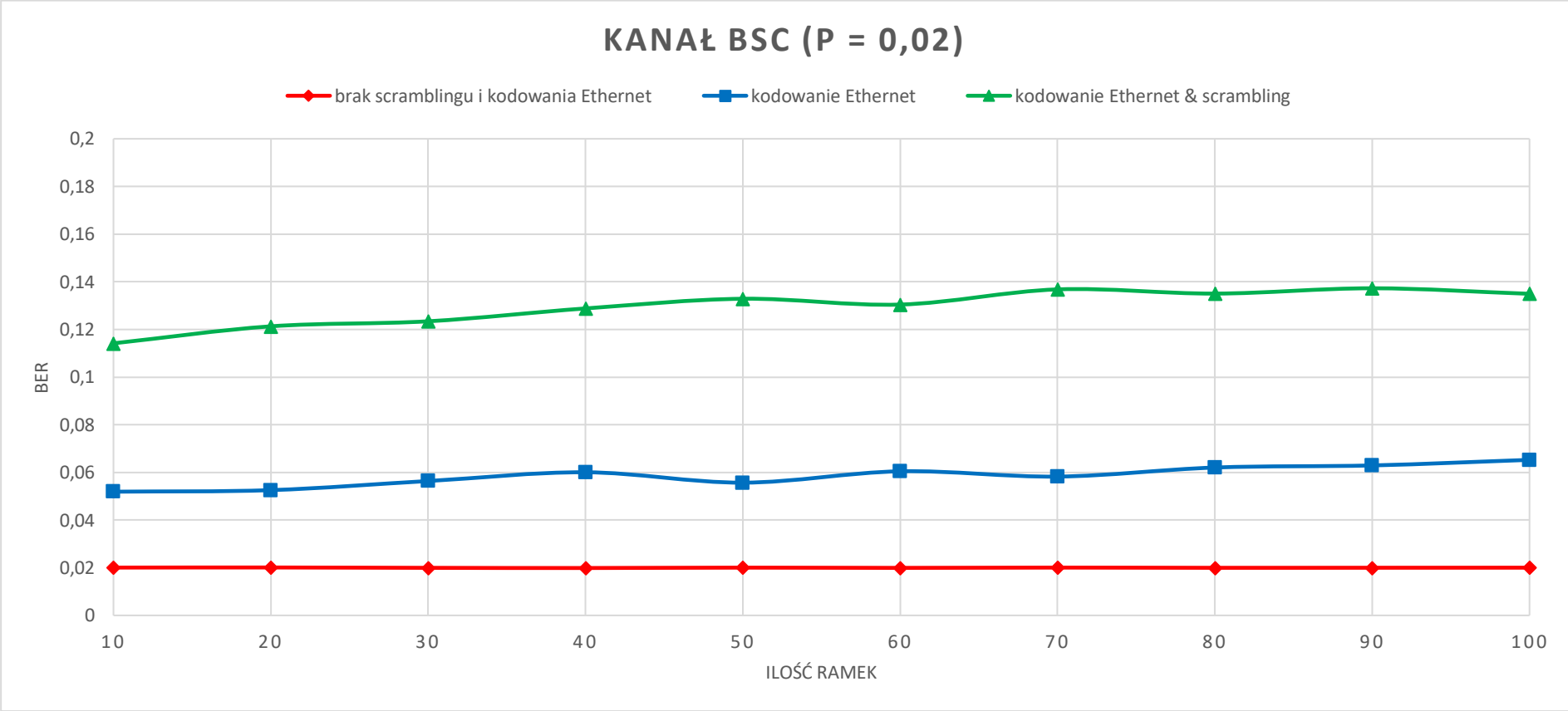
ILOŚĆ RAMEK 64b	10 (640b)	20 (1280b)	30 (1920b)	40 (2460b)	50 (3200b)	60 (3840b)	70 (4480b)	80 (5120b)	90 (5760b)	100 (6400b)
BRAK SCRAMBLINGU [BER]	0,011	0,010	0,010	0,010	0,011	0,010	0,010	0,010	0,011	0,011
ETHERNET [BER]	0,026	0,028	0,027	0,029	0,031	0,031	0,037	0,037	0,032	0,034
SCRAMBLING & ETHERNET [BER]	0,062	0,068	0,067	0,069	0,072	0,072	0,069	0,074	0,070	0,072



Wykres 3. Testy – kanał BSC z parametrem p = 0,01

3.5.4. Testy – kanał BSC z prawdopodobieństwem p = 0,02

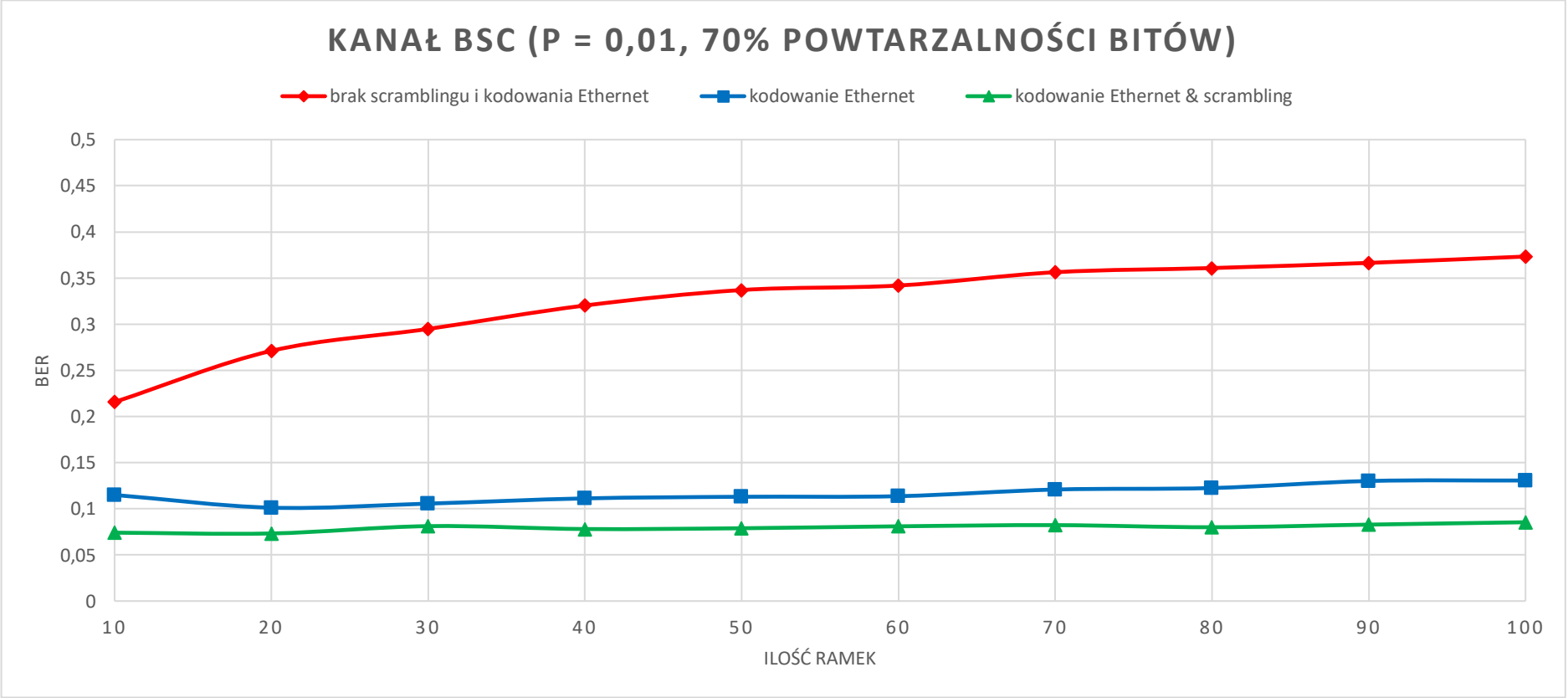
ILOŚĆ RAMEK 64b	10 (640b)	20 (1280b)	30 (1920b)	40 (2460b)	50 (3200b)	60 (3840b)	70 (4480b)	80 (5120b)	90 (5760b)	100 (6400b)
BRAK SCRAMBLINGU [BER]	0,021	0,021	0,020	0,020	0,021	0,020	0,021	0,020	0,020	0,021
ETHERNET [BER]	0,052	0,053	0,057	0,061	0,056	0,061	0,059	0,063	0,063	0,066
SCRAMBLING & ETHERNET [BER]	0,115	0,122	0,124	0,129	0,133	0,131	0,137	0,136	0,138	0,135



Wykres 4. Testy – kanał BSC z prawdopodobieństwem p = 0,02

3.5.5. Testy – kanał BSC – p = 0,01 i danymi z tendencją do powtórzeń (70%)

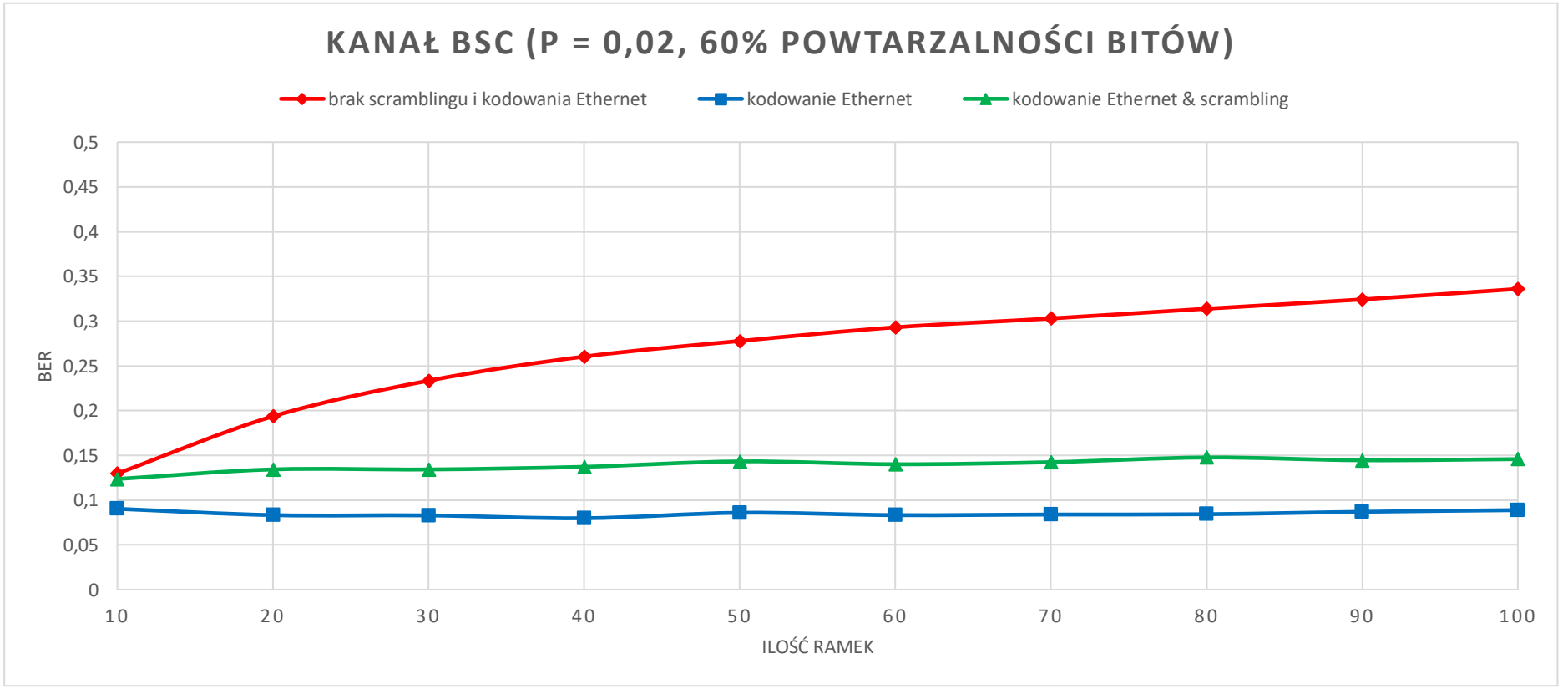
ILOŚĆ RAMEK 64b	10 (640b)	20 (1280b)	30 (1920b)	40 (2460b)	50 (3200b)	60 (3840b)	70 (4480b)	80 (5120b)	90 (5760b)	100 (6400b)
BRAK SCRAMBLINGU [BER]	0,216	0,272	0,295	0,321	0,338	0,342	0,357	0,361	0,367	0,374
ETHERNET [BER]	0,115	0,102	0,106	0,112	0,113	0,114	0,121	0,123	0,131	0,131
SCRAMBLING & ETHERNET [BER]	0,075	0,074	0,082	0,079	0,079	0,081	0,083	0,080	0,083	0,086



Wykres 5. Testy – kanał BSC z prawdopodobieństwem p = 0,01 i danymi z tendencją do powtórzeń (70%)

3.5.6. Testy – kanał BSC – p = 0,02 i danymi z tendencją do powtórzeń (60%)

ILOŚĆ RAMEK 64b	10 (640b)	20 (1280b)	30 (1920b)	40 (2460b)	50 (3200b)	60 (3840b)	70 (4480b)	80 (5120b)	90 (5760b)	100 (6400b)
BRAK SCRAMBLINGU [BER]	0,130	0,194	0,234	0,261	0,278	0,294	0,303	0,314	0,325	0,336
ETHERNET [BER]	0,091	0,084	0,083	0,080	0,086	0,084	0,084	0,085	0,087	0,089
SCRAMBLING & ETHERNET [BER]	0,124	0,135	0,135	0,138	0,144	0,141	0,143	0,148	0,145	0,146



Wykres 6. Testy – kanał BSC z prawdopodobieństwem p = 0,02 i danymi z tendencją do powtórzeń (60%)

4. Wnioski

Kodowanie Ethernet:

1. Kodowanie Ethernetowe nawet przy zastosowaniu bardzo prostego, naiwnego algorytmu korzystnie wpływa na transmisję poprzez kanał, w którym występuje desynchronizacja. Na wykresach odnotowano znaczny spadek BER przy zastosowaniu samego kodowania Ethernet.

2. Dla kanału Binary Symmetric Channel, w którym nie występuje zjawisko desynchronizacji kodowanie Ethernet okazało się być zbędne. Co więcej, zastosowanie takiego kodowania skutkuje zwiększeniem ilości błędów w transmisji ze względu na możliwe zakłamania preambuł, które skutkują w wykryciu przez dekoder fałszywej desynchronizacji oraz następnie przesunięcie prawidłowego sygnału – co z kolei skutkuje zwielokrotnieniem błędu.

Scrambling:

1. Scrambling sygnału z założenia powinien poprawiać jego parametry (mniejsza ilość ciągów tych samych symboli), czyli randomizować sygnał. Jednak z testów wynika, że jeżeli podamy na jego Wejście sygnał, który jest zrandomizowany, próba poprawienia jego parametrów tak naprawdę nieznacznie pogorszy jakość sygnału. Zjawisko to obrazują wykres [Wykres 1] oraz [Wykres 2]. Na pierwszym z nich przesyłamy sygnał ze skłonnością do powtórzeń – słabe parametry – zastosowanie scramblingu w tym przypadku pozwoliło uzyskać trzykrotnie mniejsze BER. Na drugim wykresie sygnał o dobrych parametrach po transmisji z randomizowaniem sygnału uzyskał nieco gorsze wyniki niż przy transmisji bez użycia scramblera. W rzeczywistości jednak sygnały o dobrych parametrach (krótkie ciągi tych samych bitów) występują o wiele rzadziej, przez co randomizowanie sygnału poprawia jakość transmisji danych. Nawet jeżeli sygnał miał już dobre parametry, scrambler nie zadziała tak dobrze, jednak nadal będzie to sygnał akceptowalny w praktyce, więc użycie urządzeń randomizujących sygnał daje poprawę jakości transmisji praktycznie bez żadnych konsekwencji.

2. Dla kanału Binary Symmetric Channel bez desynchronizacji zastosowanie scramblingu podobnie jak zastosowanie Ethernetu nie przynosi pozytywnych rezultatów. Randomizowanie sygnału w żaden sposób nie wpływa na BER transmisji poprzez taki kanał, a każde zakłamanie bitu skutkuje kolejnymi trzema zakłamaniami z powodu wprowadzenia zakłamanego bitu do LFSR descramblera multiplikatywnego.

3. Randomizowanie sygnału za pomocą scramblerów ma jeszcze inną cenną zaletę, przetworzony sygnał jest na swój sposób zaszyfrowany, ponieważ odzyskać z niego oryginalny sygnał można tylko za pomocą descramblera, ale takim, w którym LFSR jest odpowiednio zainicjowany. Rejestr w tym urządzeniu ma 58 bitów, co daje 2^{59} różnych możliwych wartości początkowych rejestru przesuwnego. Trzeba zatem wiedzieć jakimi wartościami wypełnić LFSR, aby odczytać dane. Jest to korzystne zjawisko, jednak w praktyce używa się jeszcze dodatkowych zabezpieczeń.

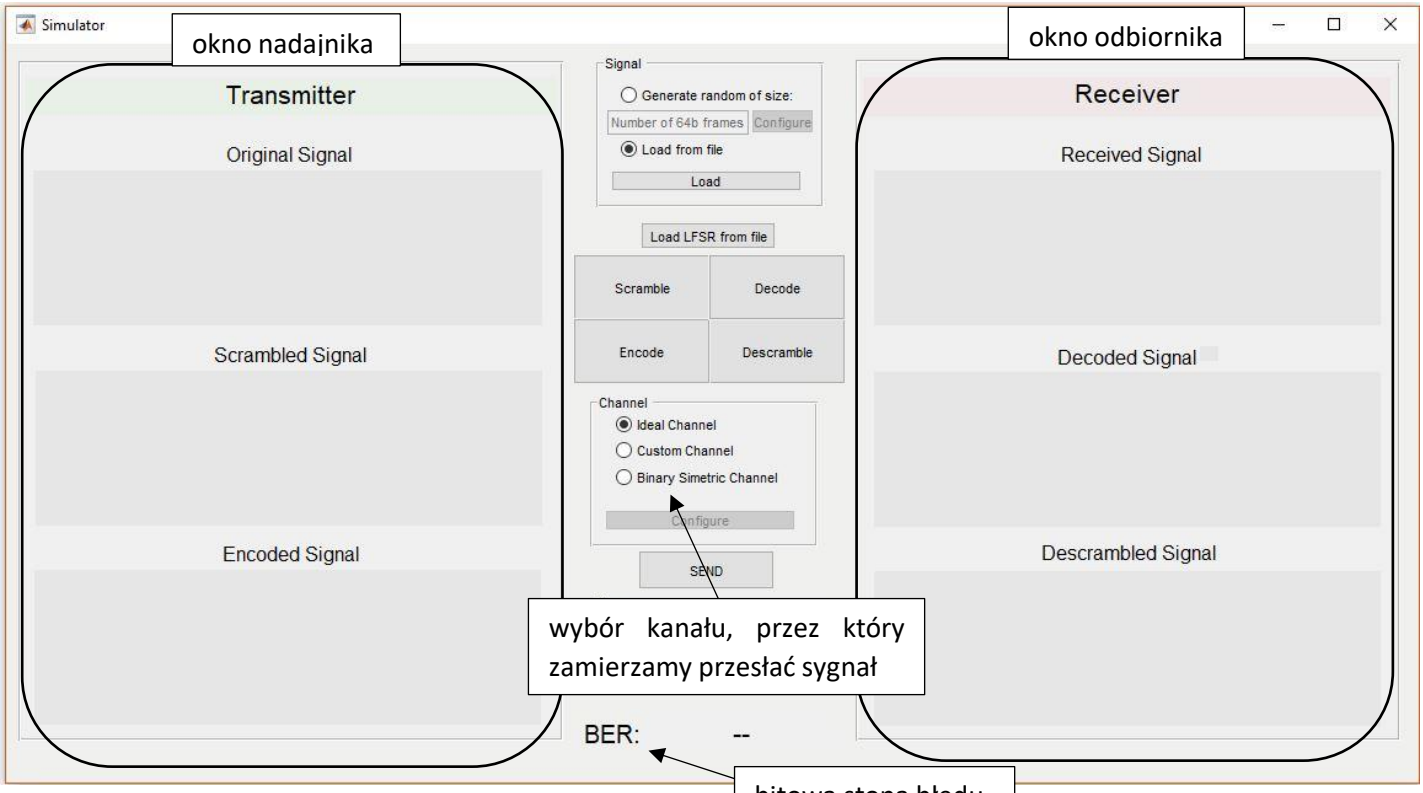
Ethernet & scrambling

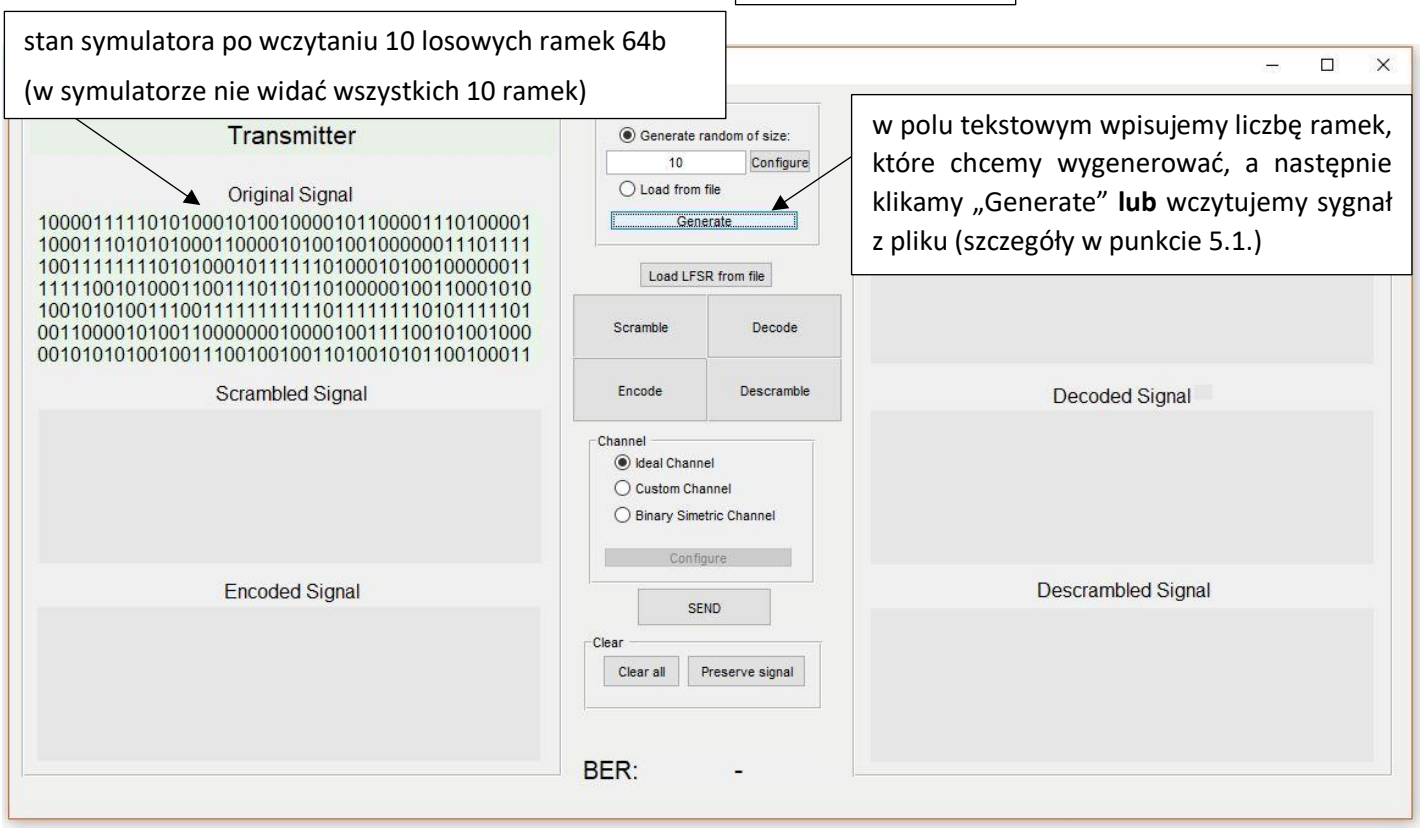
1. Jak widać na wykresie [Wykres 5] oraz [Wykres 6] nawet niewielka różnica w prawdopodobieństwie wystąpienia zakłamań podczas transmisji może drastycznie pogorszyć jakość odbieranego sygnału, wynika to z braku algorytmów wykrywających i naprawiających błędy powstałe w trakcie przesyłania danych. Każdy błędny bit generuje wtedy 3 błędne bity w końcowym sygnale.
   Można temu zapobiec poprzez:
   a. nieużywanie scramblera, co naraża nas na błędy desynchronizacji
   b. zaimplementowanie algorytmów wykrywania i naprawiania błędów transmisji, naprawiające sygnał przed descramblerem
   c. użycie scramblera addytywnego, który nie powiela błędów, ale musi mieć odświeżany rejestr LFSR co ramkę

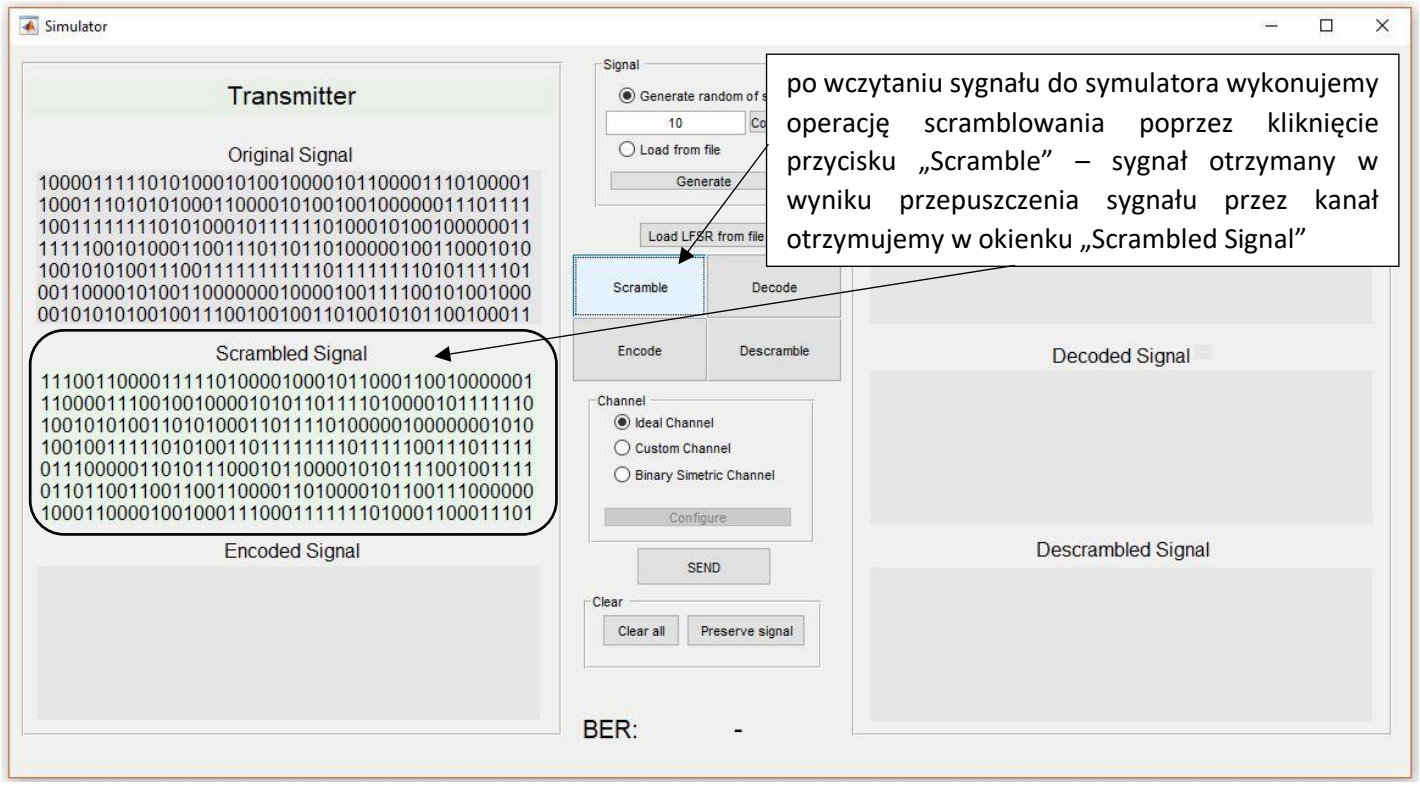
2. W kanale mieszanym [Wykres 5] oraz [Wykres 6], w którym mamy do czynienia zarówno z zakłamaniami na pojedynczych bitach jak i z desynchronizacją okazuje się, że większy wpływ na końcowy BER ma zjawisko desynchronizacji, a co za tym idzie zastosowanie kodowania Ethernet oraz scramblera daje dużo lepsze rezultaty aniżeli ich brak.

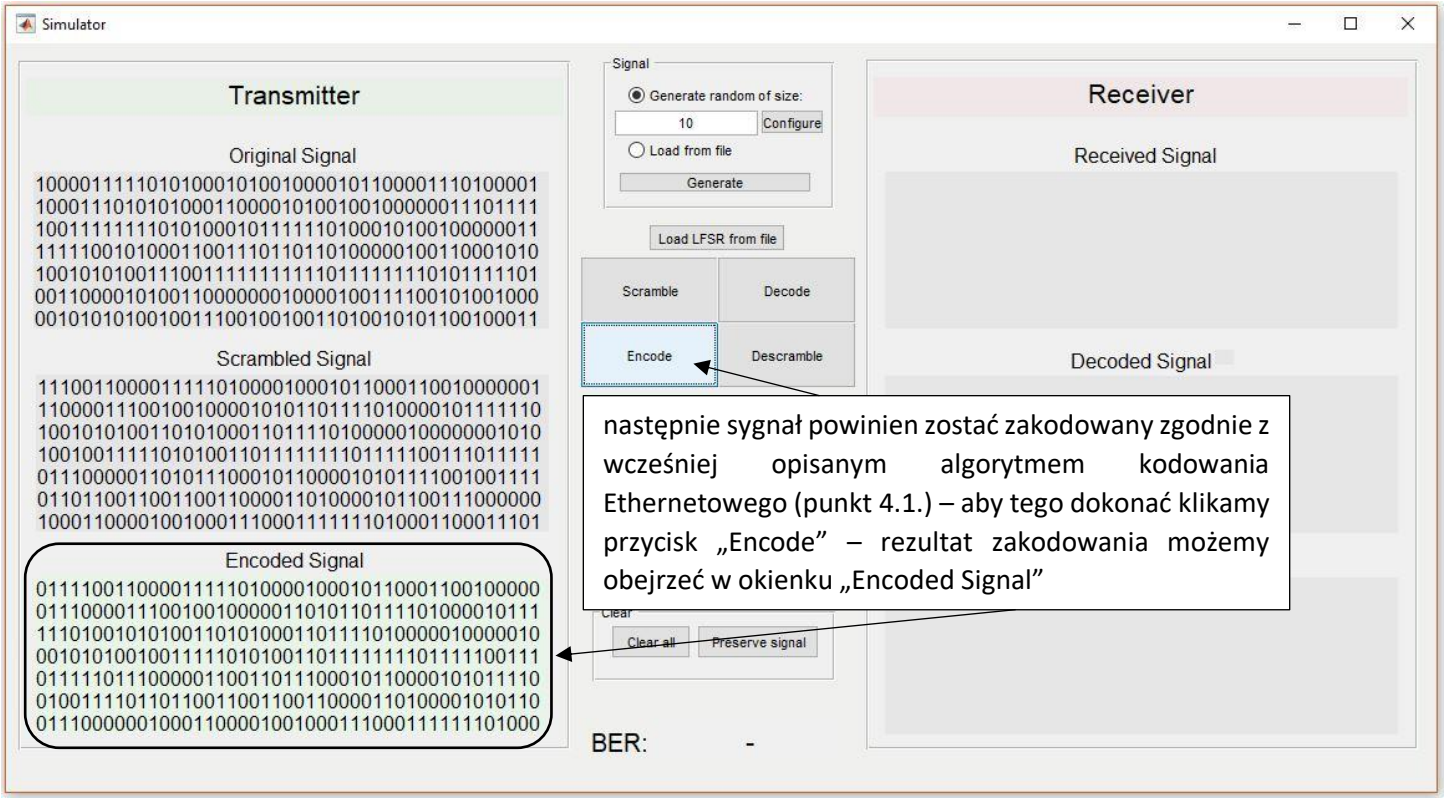
5. Symulator – opis działania

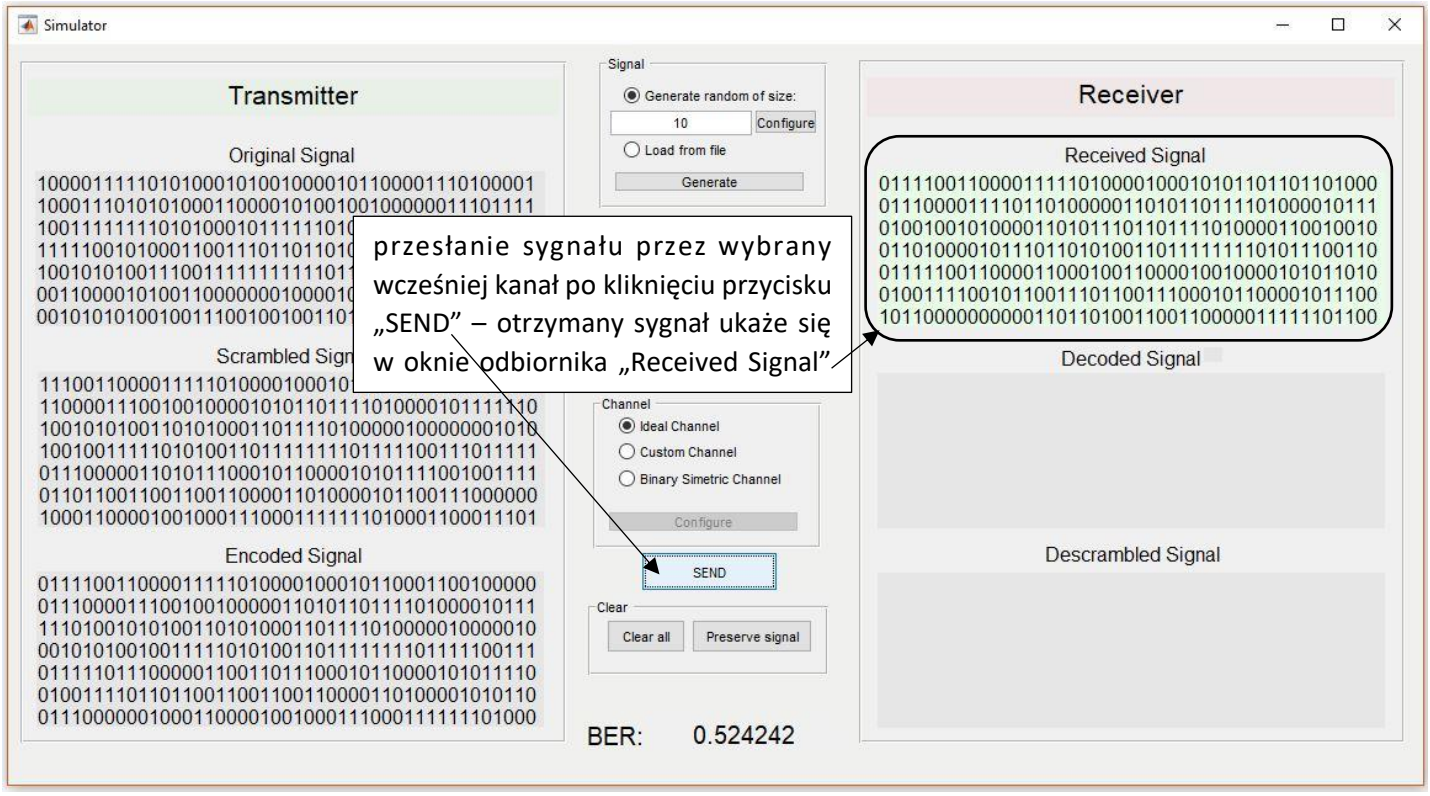
Poniższe zrzuty ekranu ukazują **przykładowy cykl działania symulatora** – zrzuty ekranu zostały opisane, w celu wyjaśnienia jego działania w poszczególnych etapach.

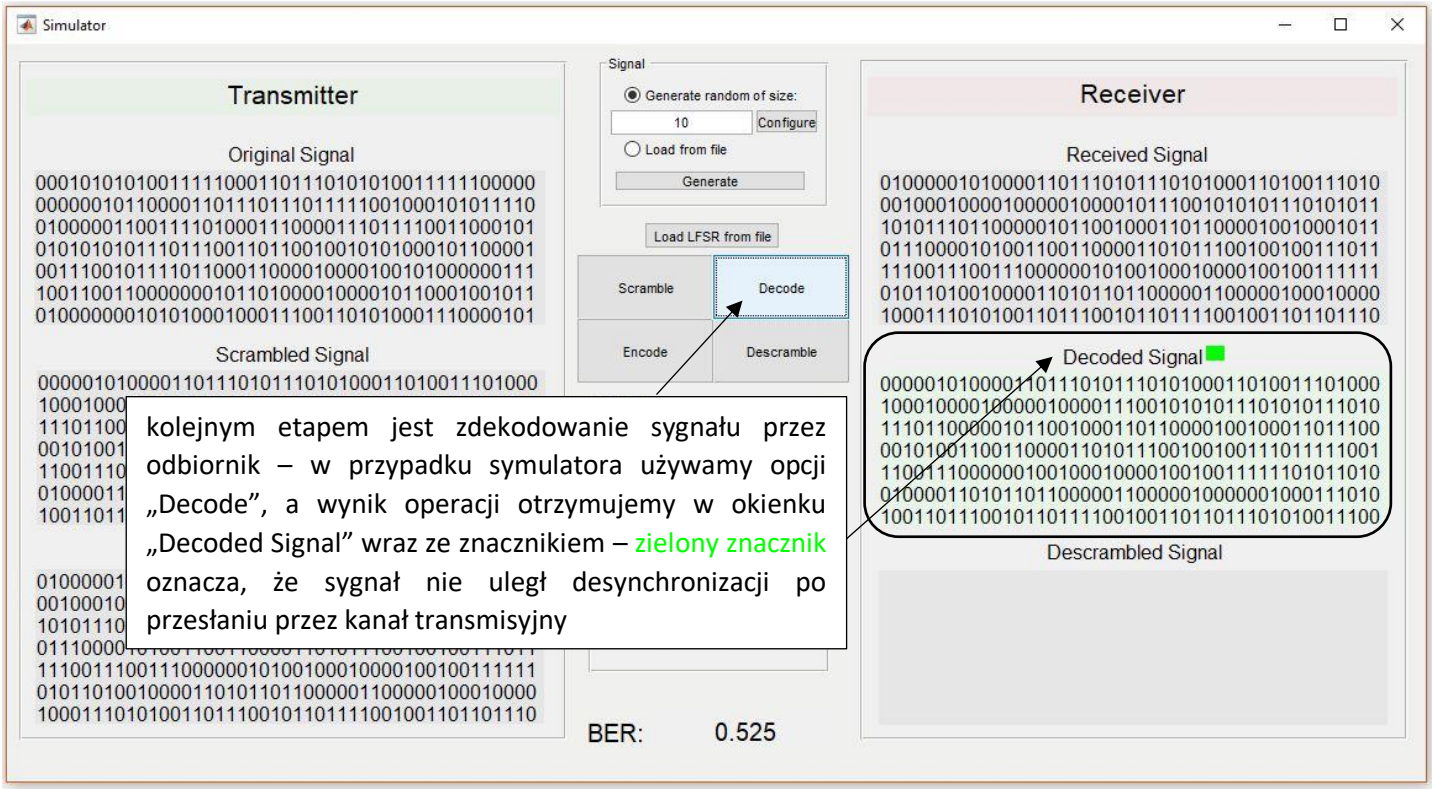


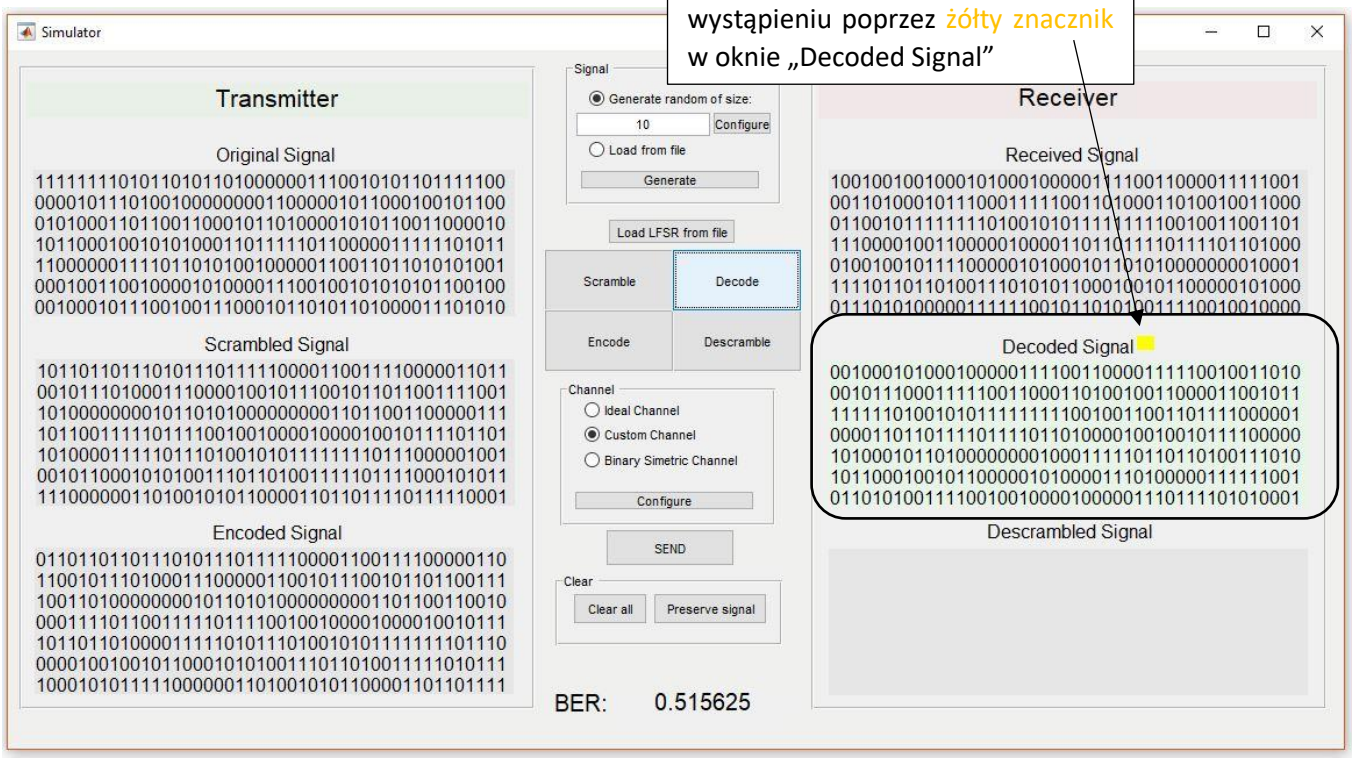


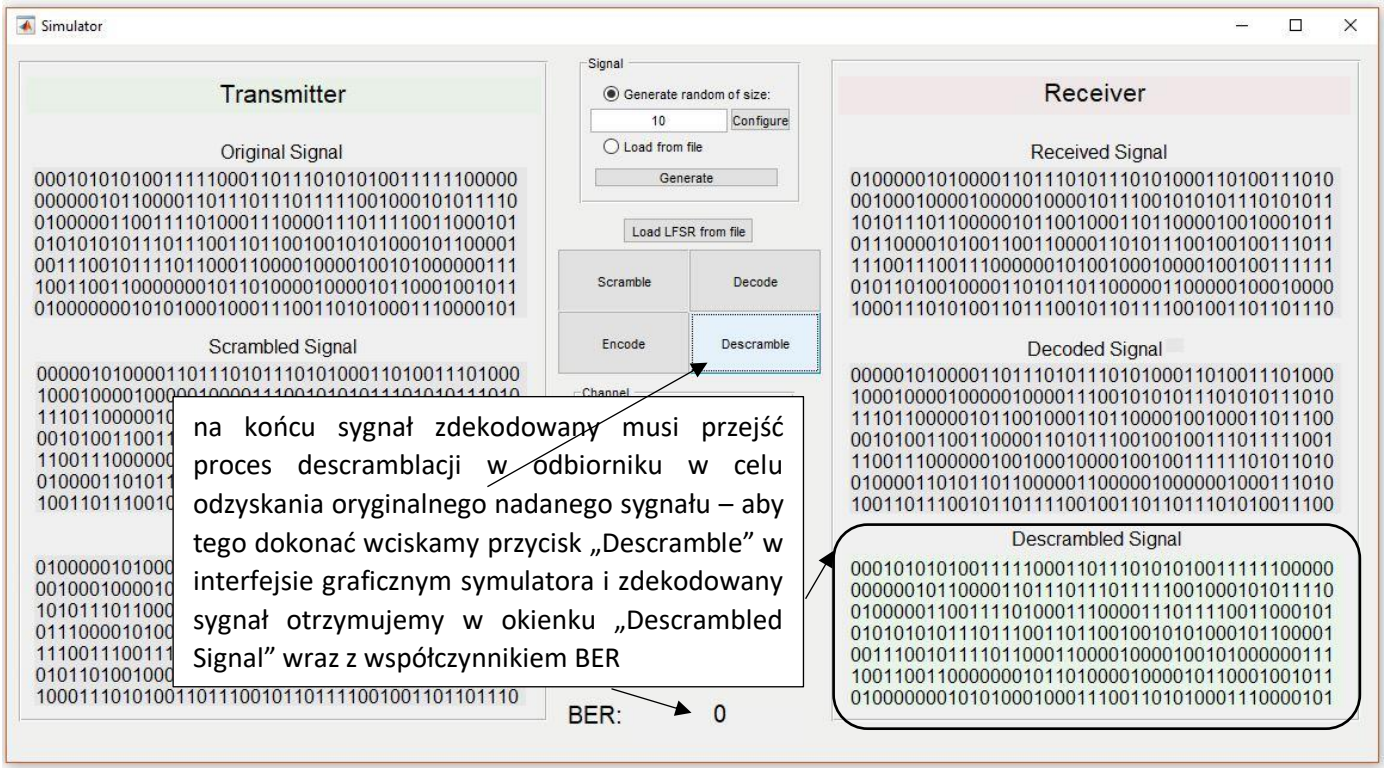

Simulator
Transmitter
Original Signal
Scrambled Signal
Encoded Signal
Signal
Generate random of s
10
Load from file
Generate
Load LFSR from file
Scramble
Decode
Encode
Descramble
Channel
Ideal Channel
Custom Channel
Binary Simetric Channel
Configure
SEND
Clear
Clear all
Preserve signal
Decoded Signal
Descrambled Signal
BER:
-
po wczytaniu sygnału do symulatora wykonujemy operację scramblowania poprzez kliknięcie przycisku „Scramble” – sygnał otrzymany w wyniku przepuszczenia sygnału przez kanał otrzymujemy w okienku „Scrambled Signal”


Simulator
Transmitter
Original Signal
Scrambled Signal
Encoded Signal
Signal
Generate random of size:
10
Configure
Load from file
Generate
Load LFSR from file
Scramble
Decode
Encode
Descramble
Clear all
Preserve signal
Receiver
Received Signal
Decoded Signal
BER:
-
następnie sygnał powinien zostać zakodowany zgodnie z wcześniej opisanym algorytmem kodowania Ethernetowego (punkt 4.1.) – aby tego dokonać klikamy przycisk „Encode” – rezultat zakodowania możemy obejrzeć w okienku „Encoded Signal”


Simulator
Transmitter
Original Signal
Scrambled Sign
Encoded Signal
Signal
Generate random of size:
10
Configure
Load from file
Generate
Channel
Ideal Channel
Custom Channel
Binary Simetric Channel
Configure
SEND
Clear
Clear all
Preserve signal
BER: 0.524242
Receiver
Received Signal
Decoded Signal
Descrambled Signal
przesłanie sygnału przez wybrany wcześniej kanał po kliknięciu przycisku „SEND” – otrzymany sygnał ukaże się w oknie odbiornika „Received Signal”


Simulator
Transmitter
Original Signal
Scrambled Signal
Signal
Generate random of size:
10
Configure
Load from file
Generate
Load LFSR from file
Scramble
Decode
Encode
Descramble
BER: 0.525
Receiver
Received Signal
Decoded Signal
Descrambled Signal
kolejnym etapem jest zdekodowanie sygnału przez odbiornik – w przypadku symulatora używamy opcji „Decode”, a wynik operacji otrzymujemy w okienku „Decoded Signal” wraz ze znacznikiem – zielony znacznik oznacza, że sygnał nie uległ desynchronizacji po przesłaniu przez kanał transmisyjny


W przypadku desynchronizacji użytkownik jest ostrzeżony o jej wystąpieniu poprzez żółty znacznik w oknie „Decoded Signal”
Simulator
Transmitter
Original Signal
Scrambled Signal
Encoded Signal
Signal
Generate random of size:
10
Configure
Load from file
Generate
Load LFSR from file
Scramble
Decode
Encode
Descramble
Channel
Ideal Channel
Custom Channel
Binary Simetric Channel
Configure
SEND
Clear
Clear all
Preserve signal
BER: 0.515625
Receiver
Received Signal
Decoded Signal
Descrambled Signal

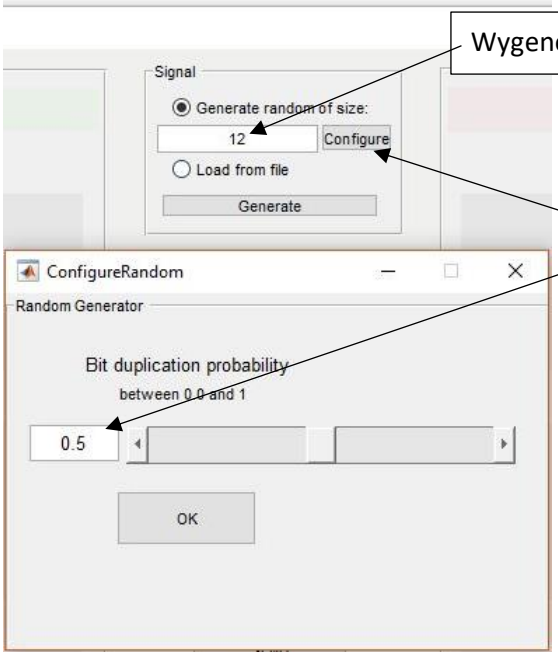

Simulator
Transmitter
Original Signal
Scrambled Signal
Signal
Generate random of size:
10
Configure
Load from file
Generate
Load LFSR from file
Scramble
Decode
Encode
Descramble
Channel
na końcu sygnał zdekodowany musi przejść proces descramblacji w odbiorniku w celu odzyskania oryginalnego nadanego sygnału – aby tego dokonać wciskamy przycisk „Descramble” w interfejsie graficznym symulatora i zdekodowany sygnał otrzymujemy w okienku „Descrambled Signal” wraz z współczynnikiem BER
BER: 0
Receiver
Received Signal
Decoded Signal
Descrambled Signal

5.1. Symulator – dane wejściowe

W symulatorze zaimplementowano dwie opcje wczytania sygnału wejściowego:

- wylosowanie sygnału za pomocą opcji symulatora
- wczytanie sygnału z pliku – zawartość pliku została ukazana poniżej w formie zrzutu ekranu – jako pierwszy parametr podawana jest długość sygnału którą chcemy wczytać, a następnie ciągi zer i jedynek, będące **sygnałem wejściowym**.

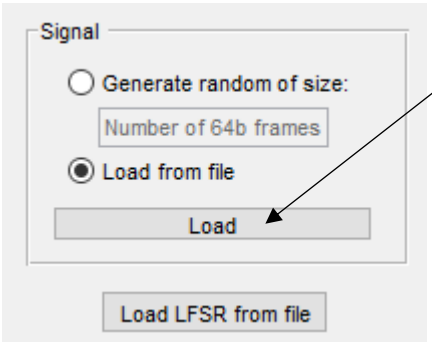
5.2. Wczytanie losowego sygnału przy użyciu opcji symulatora



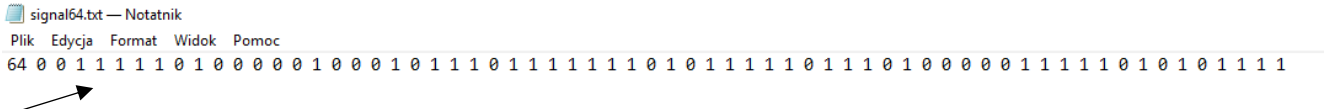
Wygenerowane zostanie 12 losowych ramek o długości 64b każda

Przycisk „Configure" pozwala ustalić z jakim prawdopodobieństwem ma wystąpić duplikacja bitów

5.3. Wczytanie sygnału z pliku tekstowego (*.txt)



Wybieramy opcję wczytania sygnału z pliku – klikamy przycisk „Load" – sygnał zostanie wczytany



Przykładowy plik tekstowy z sygnałem – pierwsza liczba to **długość sygnału w bitach** a następnie podawany jest sygnał

6. Bibliografia

[1] D. R. Stauffer, „High Speed Serdes Devices and Applications,” Springer, pp. 140-141.

[2] [Online]. Available: http://www.ece.ubc.ca/~edc/3525.jan2014/lectures/lec13.pdf.

[3] [Online]. Available: http://www.ece.ubc.ca/~edc/3525.jan2014/lectures/lec8.pdf.

[4] „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Scrambler.

[5] J. Doke, MathWorks, [Online]. Available: https://www.mathworks.com/videos/creating-a-gui-with-guide-68979.html.

[6] MathWorks, [Online]. Available: https://www.mathworks.com/help/matlab/object-oriented-programming-in-matlab.html.