# Transmisja Danych – Lab 10

Krystian Bartosik 213A, nr 44266

Kod źródłowy:

```cpp
// Krystian Bartosik
// bk44266@zut.edu.pl
//   FEDCBA
#define _USE_MATH_DEFINES
#include <iostream>
#include <string>
#include <fstream>
#include "math.h"
#include <complex>
#include <cstddef>
#include <bitset>
#include <stdlib.h>

using namespace std;

string S2BS(const char* s, string Endian)
{
    string result = "";

    if (Endian == "BigEndian")
    {
        for (int j = 0; j < strlen(s); j++)
        {
            result = result + bitset<8>(s[j]).to_string();
        }
    }
    // "test" = 01110100011001010111001101110100

    if (Endian == "LittleEndian")
    {
        for (int j = 0; j < strlen(s); j++)
        {
            result = bitset<8>(s[j]).to_string() + result;
        }
    }

    //cout << result << endl;
    return result;
}

char SB2S(string S)
{
    int X = stoi(S, nullptr, 2);
    return X;
}
```

```
int** G_Multiply(int** D) // Przyjmuje 4bitową tablicę
{
    int G[7][4] = { {1,1,0,1},{1,0,1,1},{1,0,0,0},{0,1,1,1},{0,1,0,0},{0,0,1,0},{0,0,0,1} };
    int** C = new int* [8];
    int BitKontrolny = 0;

    for (int j = 0; j < 8; j++)
        C[j] = new int[1];

    for (int j = 0; j < 8; j++)
    {
        C[j][0] = 0;
    }

    for (int j = 0; j < 7; j++)
    {
        for (int i = 0; i < 4; i++)
        {
            //cout << C[j][0] << "+(" << D[0][i]
            C[j][0] = C[j][0] + (D[i][0] * G[j][i]);
        }
        C[j][0] = C[j][0] % 2;
        BitKontrolny = BitKontrolny + C[j][0];
    }

    C[7][0] = BitKontrolny % 2;

    return C;
}

string Hamming_Nadajnik(string S)
{
    cout << "[NAD] Dane do wyslania:  " << S << endl;
    int** C = new int* [4];

    for (int j = 0; j < 4; j++)
    {
        C[j] = new int[1];
    }

    for (int j = 0; j < 4; j++)
    {
        C[j][0] = (int)S[j] - '0'; // Konwersja na liczbę
    }
```

```
    int** Result = G_Multiply(C);
    string X = "00000000";

    for (int j = 0; j < 8; j++)
    {
        X[j] = Result[j][0] + 48; // Konwersja na znak
    }
    cout << "[NAD] Nadana wiadomosc:  " << X << endl;
    return X;
}

void Zegar(double Czestotliwosc, double Probkowanie)
{
    fstream File1;
    File1.open("C:/Users/Qrystian/Desktop/results1.txt", ios::out);

    int Check = 0;
    int Bit = 1;
    for (double j = 0; j < Czestotliwosc; j = j + Probkowanie)
    {
        File1 << j << " " << Bit << endl;

        if (Check == 500)
        {
            Check = 0;
            if (Bit == 0)
                Bit = 1;
            else
                Bit = 0;
            }

        Check++;
    }

    File1.close();
}

double TTL(char t)
{
    double tt = (double)t - '0'; // Konwersja na liczbę

    if (tt == 0)
    {
        return 0.0;
```

```
    }

    if (tt == 1)
    {
        return 1.0;
    }
}

double zA(double A1, double A2, double f, double Fi, char T, double t)
{
    double tt = (double)T - '0'; // Konwersja na liczbę
    if (tt == 0)
    {
        return A1 * sin(2.0 * M_PI * f * t + Fi);
    }

    if (tt == 1)
    {
        return A2 * sin(2.0 * M_PI * f * t + Fi);
    }
}

double zF(double A, long double f0, double f1, double Fi, char T, double t)
{
    double tt = (double)T - '0'; // Konwersja na liczbę
    if (tt == 0)
    {
        return A * sin(2.0 * M_PI * f0 * t + Fi);
    }

    if (tt == 1)
    {
        return A * sin(2.0 * M_PI * f1 * t + Fi);
    }
}

double zP(double A, double f, double Fi1, double Fi2, char T, double t)
{
    double tt = (double)T - '0'; // Konwersja na liczbę
    if (tt == 0)
    {
        return A * sin(2.0 * M_PI * f * t + Fi1);
    }
```

```
    if (tt == 1)
    {
        return A * sin(2.0 * M_PI * f * t + Fi2);
    }
}

string Hamming_Odbiornik(string S)
{
    cout << endl;
    cout << "[ODB] Odebrana wiadomosc:" << S << endl;
    int H[3][7] = { {0,0,0,1,1,1,1},{0,1,1,0,0,1,1},{1,0,1,0,1,0,1} };
    int** C = new int* [8];
    int** Answer = new int* [3];
    int BitParzystosci = 0;

    for (int j = 0; j < 8; j++)
    {
        C[j] = new int[1];
        C[j][0] = (int)S[j] - '0';
        BitParzystosci = BitParzystosci + C[j][0];
    }

    BitParzystosci = BitParzystosci % 2;
    cout << "[ODB] Bit parzystosci:   " << BitParzystosci << endl;

    for (int j = 0; j < 3; j++)
    {
        Answer[j] = new int[1];
        Answer[j][0] = 0;
    }

    cout << "[ODB] Odkodowana macierz:";

    for (int j = 0; j < 3; j++)
    {
        for (int i = 0; i < 7; i++)
        {
            Answer[j][0] = Answer[j][0] + (C[i][0] * H[j][i]);
        }
        Answer[j][0] = Answer[j][0] % 2;
        cout << Answer[j][0];
    }
    cout << endl;
```

```
    bool Poprawnosc;
    for (int j = 0; j < 3; j++)
    {
        if (Answer[j][0] == 0)
        {
            Poprawnosc = true;
            continue;
        }
        else
        {
            Poprawnosc = false;
            break;
        }
    }

    if ((Poprawnosc == true) && (BitParzystosci == 1))
    {
        cout << "[ODB] Pojedynczy blad na pozycji 8!" << endl;
        C[7][0] = 0;
        BitParzystosci = 0;
        cout << "[ODB] Blad naprawiony, przekazuje dalej!" << endl;
    }

    if ((Poprawnosc == true) && (BitParzystosci == 0))
    {
        cout << "[ODB] Dane poprawne!" << endl;
        string X = "0000";

        X[0] = C[2][0] + 48;
        X[1] = C[4][0] + 48;
        X[2] = C[5][0] + 48;
        X[3] = C[6][0] + 48;

        cout << "[ODB] Odebrane dane:     " << X << endl << endl;
        return X;
    }

    if ((Poprawnosc == false) && (BitParzystosci == 1))
    {
        cout << "[ODB] Pojedynczy blad!" << endl;
        int Pozycja = 0;
        for (int j = 0; j < 3; j++)
        {
            Pozycja = Pozycja + (Answer[2 - j][0] * pow(2, j));
```

```
        }

        cout << "[ODB] Blad na pozycji:   " << Pozycja - 1 << endl;
        if (C[Pozycja - 1][0] == 1)
            C[Pozycja - 1][0] = 0;
        else
            C[Pozycja - 1][0] = 1;

        cout << "[ODB] Blad naprawiony!" << endl;

        string X = "0000";

        X[0] = C[2][0] + 48;
        X[1] = C[4][0] + 48;
        X[2] = C[5][0] + 48;
        X[3] = C[6][0] + 48;

        cout << "[ODB] Odebrane dane:     " << X << endl << endl;
        return X;
    }

    if ((Poprawnosc == false) && (BitParzystosci == 0))
    {
        cout << "[ODB] Podwojny blad!" << endl;
        cout << "[ODB] Pakiet odrzucony!" << endl << endl;
        return "Pakiet uszkodzony";
    }
}

double Szum(double Alpha)
{
    return ((rand() % 10) / 10.0) * (1 - (Alpha));
}

complex<double>* DFT(double* Tab, int n)
{
    complex<double>* c = new complex<double>[n];
    complex<double> i = 0.0 + 1.0i;
    for (int k = 0; k < n; k++)
    {
        c[k] = 0.0 + 0.0i;

        for (int j = 0; j < n; j++)
        {
            c[k] = c[k] + (Tab[j] * exp(-2 * M_PI * i * (double)k * (double)j / double(n)));
```

```
        }
    }
    return c;
}

int main()
{
    const char* Napis = "a";
    string S = S2BS(Napis, "BigEndian");
    cout << "Napis: " << Napis << endl << "Ciag binarny: " << S << endl;

    string SSS = "0000000000000000";

    int Poz = 0;
    for (int k = 0; k < S.length(); k++)
    {
        if (k % 4 == 0)
        {
            string SS = S.substr(Poz, 4);
            SS = Hamming_Nadajnik(SS);
            cout << endl;

            for (int a = 0; a < 8; a++)
            {
                if (Poz == 0)
                    SSS[a] = SS[a];
                else
                    SSS[a + 8] = SS[a];
            }

            Poz = Poz + 4;
        }
    }

    cout << "Napis po kodowaniu: " << SSS << endl;
    S = SSS;

    fstream File1;
    File1.open("C:/Users/Qrystian/Desktop/results1.txt", ios::out);
    double* Tab;
    double* M;
    double* Y;
```

```
complex<double>* X;

int size = (unsigned int)(S.length() / 0.001);
Tab = new double[size];
M = new double[size];

//Zegar(S.length(), 0.001);
double Suma = 0;
int Checker = 0;

string SS = "0000000000000000";
int Sj = 0;
int jj = 0;
double Alfa = 0.001;

for (double t = 0; t < S.length(); t = t + 0.001)
{
    // ASK
    /*
    //File1 << t << " " << zA(0.0, 1.0, ((double)S.length()/0.001) * pow(1000, -1), 2 * M_PI, S[floor(t)], t) << endl; //
    Tab[jj] = (zA(0.0, 1.0, ((double)S.length() / 0.001) * pow(1000, -1), 2 * M_PI, S[floor(t)], t) * Alfa) + Szum(Alfa);
    //File1 << t << " " << Tab[jj] << endl; //Po szumie
    jj = jj + 1;


    double Y = (zA(0.0, 1.0, ((double)S.length() / 0.001) * pow(1000, -1), 2 * M_PI, S[floor(t)], t) * Alfa + Szum(Alfa))
    //File1 << t << " " << Y << endl; //do dekodowania


    if (Checker == 999) // Co 1000 próbek = co jeden bit
    {
        for (double j = 0.0 + t; j < 1 + t; j = j + 0.01)
        {
            if (Suma > 0.3)   // h=0.3, odzyskanie sygnału
            {
                SS[Sj] = '1';
                Sj++;
                break;
            }
            else
            {
                SS[Sj] = '0';
```

```
            Sj++;
            break;
        }
    }
    Suma = 0;
    Checker = 0;
}
else
{
    Suma = Suma + abs(Y); // Sumowanie kolejnych próbek = drugi krok całkowania
    Checker++;
}
*/

// PSK
/*
//File3 << t << " " << zP(1.0, ((double)S.length() / 0.01) * pow(1000, -1), 0.0, M_PI, S[floor(t)], t) << endl;
Tab[jj] = (zP(1.0, ((double)S.length() / 0.01) * pow(1000, -1), 0.0, M_PI, S[floor(t)], t) * Alfa) + Szum(Alfa);
jj = jj + 1;

double Y = (zP(1.0, ((double)S.length() / 0.01) * pow(1000, -1), 0.0, M_PI, S[floor(t)], t) * Alfa + Szum(Alfa))

if (Checker == 999)
{
    for (double j = 0.0 + t; j < 1 + t; j = j + 0.01)
    {
        if (Suma > 0)   // h=0, odzyskanie sygnału
        {
            SS[Sj] = '1';
            Sj++;
            break;
        }
        else
        {
            SS[Sj] = '0';
            Sj++;
            break;
        }
    }
    Suma = 0;
    Checker = 0;
}
else
{
    Suma = Suma + Y;
```

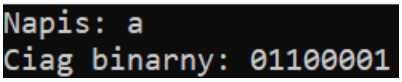
```
        Checker++;
    }
    */


    // FSK

    Tab[jj] = (zF(1.0, 1.0, 5.0, 2 * M_PI, S[floor(t)], t) * Alfa) + Szum(Alfa); // Do dft

    double x1 = ((zF(1.0, 1.0, 5.0, 2 * M_PI, S[floor(t)], t) * Alfa) + Szum(Alfa)) * ((zF(1.0, 1.0, 1.0, 2 * M_PI, S[floor(t)], t) * Alfa) + Szum(Alfa));
    double x2 = ((zF(1.0, 1.0, 5.0, 2 * M_PI, S[floor(t)], t) * Alfa) + Szum(Alfa)) * ((zF(1.0, 5.0, 5.0, 2 * M_PI, S[floor(t)], t) * Alfa) + Szum(Alfa));
    double Y = (x2 * 0.01) - (x1 * 0.01);

    if (Checker == 999)
    {
        for (double j = 0.0 + t; j < 1 + t; j = j + 0.01)
        {
            if (Suma > 0)   // h=0, odzyskanie sygnału
            {
                SS[Sj] = '1';
                Sj++;
                break;
            }
            else
            {
                SS[Sj] = '0';
                Sj++;
                break;
            }

        }
        Suma = 0;
        Checker = 0;
    }
    else
    {
        Suma = Suma + Y;
        Checker++;
    }

}
```

```
cout << "Sygnal po demodula: " << SS << endl;
int j = 0;

for (double t = 0; t < S.length(); t = t + 0.001)
{
    //Tab[j] = zA(0.0, 1.0, ((double)S.length() / 0.001) * pow(1000, -1), 2 * M_PI, S[floor(t)], t);
    Tab[j] = (zA(0.0, 1.0, ((double)S.length() / 0.001) * pow(1000, -1), 2 * M_PI, S[floor(t)], t) * Alfa) + Szum(Alfa);

    //Tab[j] = zP(1.0, ((double)S.length() / 0.01) * pow(1000, -1), 0.0, M_PI, S[floor(t)], t);
    //Tab[j] = zF(1.0, 1.0, 5.0, 2 * M_PI, S[floor(t)], t);
    j++;
}

size = (unsigned int)(S.length() / 0.001);
X = DFT(Tab, size);

for (j = 0; j < size; j++)
{
    M[j] = sqrt(pow(X[j].real(), 2) + pow(X[j].imag(), 2));
    M[j] = 10 * log10(M[j]);
}

j = 0;
for (double t = 0.0; t <= S.length(); t = t + 0.001)
{
    File1 << t << " " << M[j] << endl;
    j++;
}

string S1 = SS.substr(0, 8);
string S2 = SS.substr(8, 8);
string SS1 = "";
string SS2 = "";

SS1 = Hamming_Odbiornik(S1);
SS2 = Hamming_Odbiornik(S2);

string Sx = "";
Sx.append(SS1);
Sx.append(SS2);
```

```
cout << "Odebrany napis: " << Sx << endl;

cout << "Odebrana wiadomosc: " << SB2S(Sx);
```

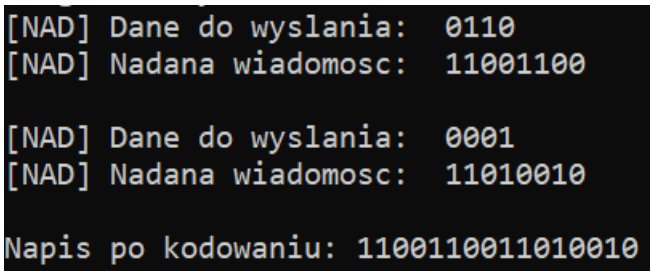
# Przebieg

Do zakodowania został wybrany napis „a”. Jego ciąg binarny to 01100001:




*Rysunek 1*
*Napis i ciąg binarny*

Ciąg binarny został zakodowany SECDED. Jako że jest to 8 bitów, kodowane były najpierw pierwsze 4, później następne. Uzyskano dzięki temu 16bitowy ciąg:




*Rysunek 2*
*Zakodowany napis*

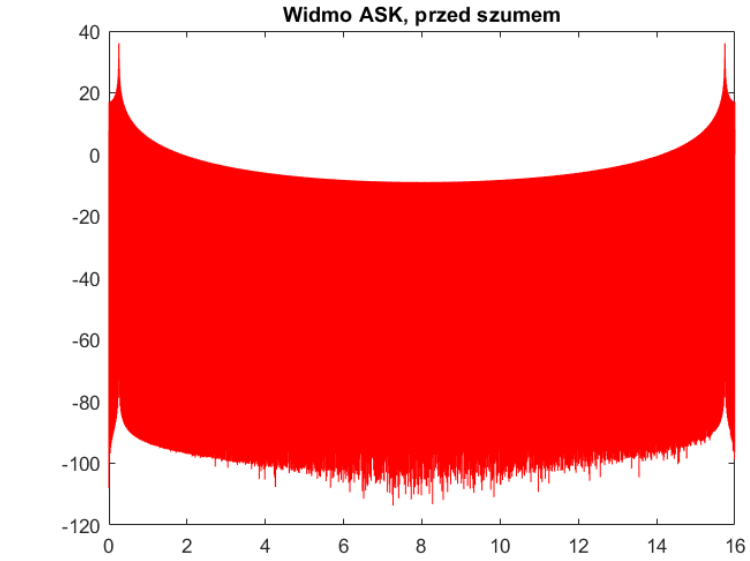
Następnie ten 16bitowy ciąg został kolejno zmodulowany ASK, FSK, PSK. Poniżej wykresy wszystkich, z różnymi parametrami:

## ASK

Dla ASK zostały znalezione 3 wartości alfa:

- Sporadyczne błędy: 0.886
- Częste błędy: 0.883
- Bardzo częste błędy: 0.88

Widmo bez szumu prezentuje się następująco:

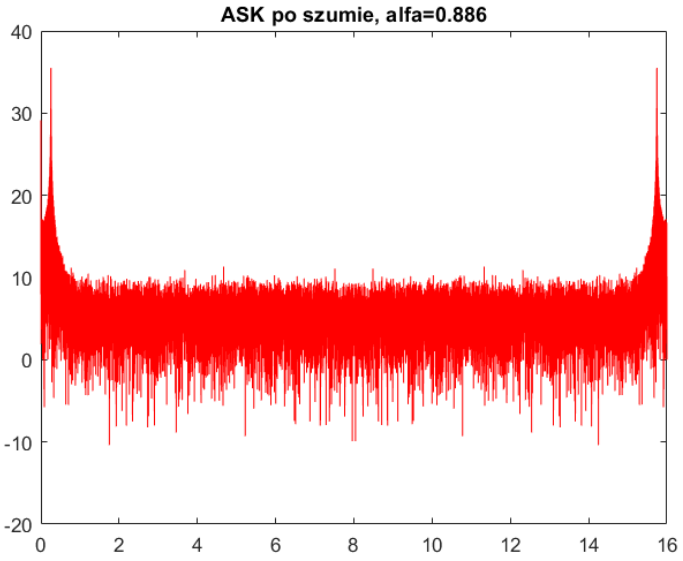



*Rysunek 3*
*Wykres widma dla ASK przed szumem*

Poniżej wykresy oraz ciągi binarne dla kolejnych wartości alfa:

```
Napis po kodowaniu: 1100110011010010
Sygnal po demodula: 1110110011010010

[ODB] Odebrana wiadomosc:11101100
[ODB] Bit parzystosci:   1
[ODB] Odkodowana macierz:011
[ODB] Pojedynczy blad!
[ODB] Blad na pozycji:   2
[ODB] Blad naprawiony!
[ODB] Odebrane dane:     0110


[ODB] Odebrana wiadomosc:11010010
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:000
[ODB] Dane poprawne!
[ODB] Odebrane dane:     0001

Odebrany napis: 01100001
Odebrana wiadomosc: a
```

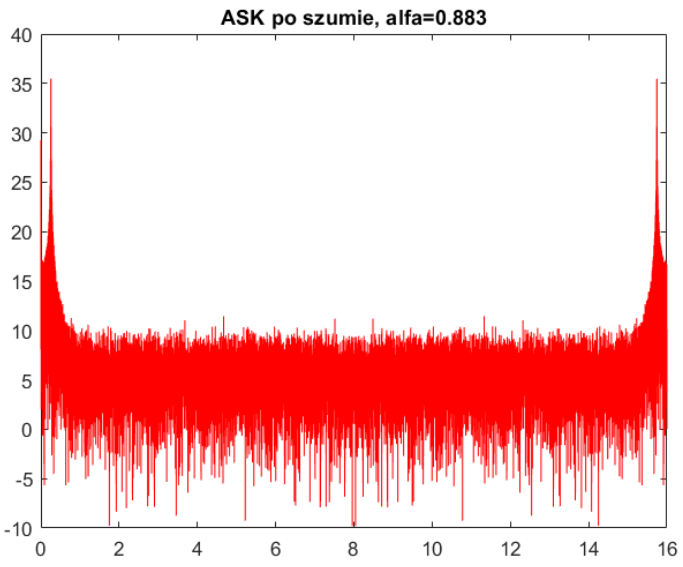
*Rysunek 4*
*Ciągi binarne dla sporadycznych błędów*




*Rysunek 5*
*Wykres dla sporadycznych błędów*

```
Napis po kodowaniu: 1100110011010010
Sygnal po demodula: 1110110111011110

[ODB] Odebrana wiadomosc:11101101
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:011
[ODB] Podwojny blad!
[ODB] Pakiet odrzucony!


[ODB] Odebrana wiadomosc:11011110
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:011
[ODB] Podwojny blad!
[ODB] Pakiet odrzucony!

Odebrany napis: Pakiet uszkodzony
```

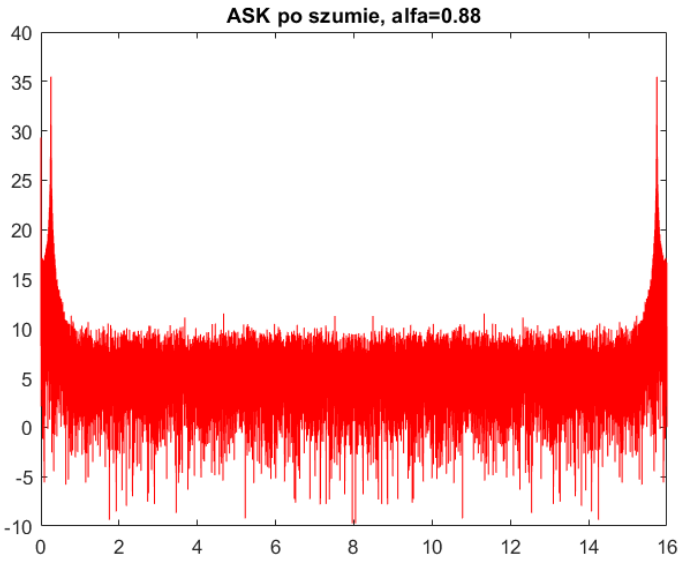
*Rysunek 6*
*Ciągi binarne dla częstych błędów*




*Rysunek 7*
*Wykres dla częstych błędów*

```
Napis po kodowaniu: 1100110011010010
Sygnal po demodula: 1111110111011110

[ODB] Odebrana wiadomosc:11111101
[ODB] Bit parzystosci:   1
[ODB] Odkodowana macierz:111
[ODB] Podwojny blad!
[ODB] Pakiet odrzucony!


[ODB] Odebrana wiadomosc:11011110
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:011
[ODB] Podwojny blad!
[ODB] Pakiet odrzucony!
```

*Rysunek 8*
*Ciągi binarne dla bardzo częstych błędów*



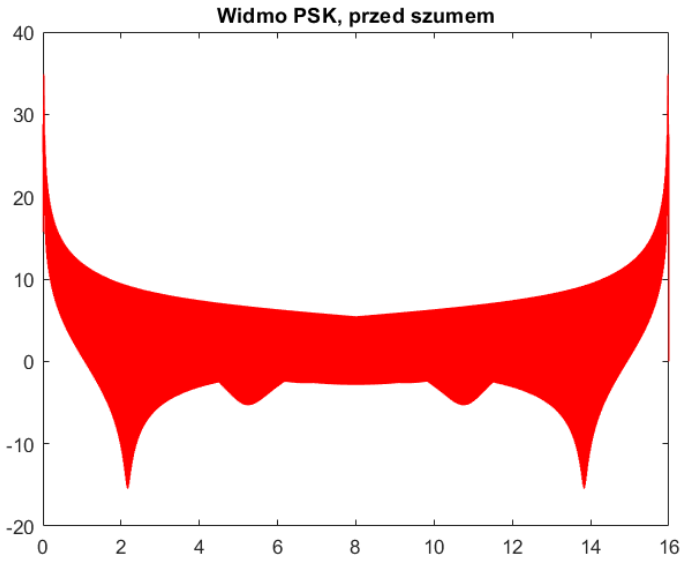

*Rysunek 9*
*Wykres dla bardzo częstych błędów*

## PSK

Dla PSK zostały znalezione 3 wartości alfa:

- Sporadyczne błędy: 0.39
- Częste błędy: 0.38
- Bardzo częste błędy: 0.37

Widmo bez szumu prezentuje się następująco:

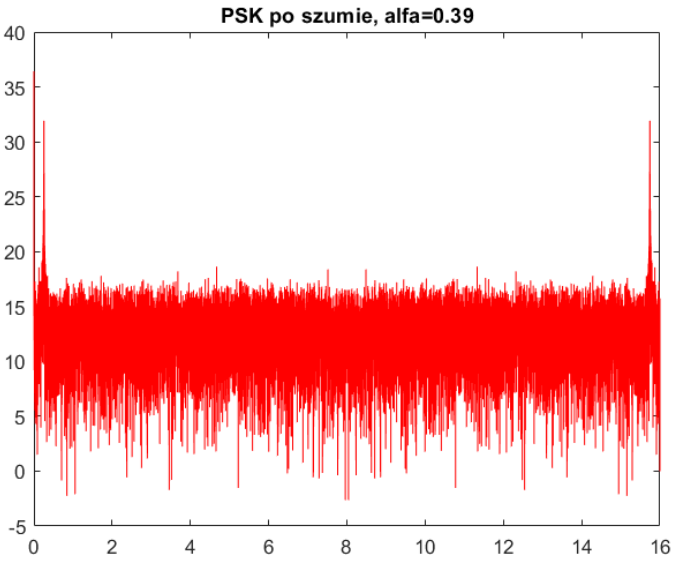



*Rysunek 10*
*Wykres widma dla PSK przed szumem*

Poniżej wykresy oraz ciągi binarne dla kolejnych wartości alfa:

```
Napis po kodowaniu: 1100110011010010
Sygnal po demodula: 1100110011010011

[ODB] Odebrana wiadomosc:11001100
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:000
[ODB] Dane poprawne!
[ODB] Odebrane dane:     0110


[ODB] Odebrana wiadomosc:11010011
[ODB] Bit parzystosci:   1
[ODB] Odkodowana macierz:000
[ODB] Pojedynczy blad na pozycji 8!
[ODB] Blad naprawiony, przekazuje dalej!
[ODB] Dane poprawne!
[ODB] Odebrane dane:     0001

Odebrany napis: 01100001
Odebrana wiadomosc: a
```

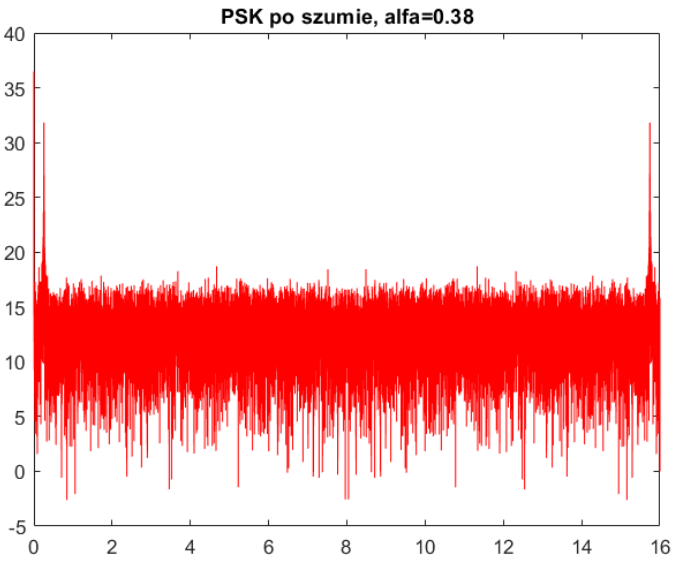
*Rysunek 11*
*Ciągi binarne dla sporadycznych błędów*




*Rysunek 12*
*Wykres dla sporadycznych błędów*

```
Napis po kodowaniu: 1100110011010010
Sygnal po demodula: 1101111011110111

[ODB] Odebrana wiadomosc:11011110
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:011
[ODB] Podwojny blad!
[ODB] Pakiet odrzucony!


[ODB] Odebrana wiadomosc:11110111
[ODB] Bit parzystosci:   1
[ODB] Odkodowana macierz:101
[ODB] Pojedynczy blad!
[ODB] Blad na pozycji:   4
[ODB] Blad naprawiony!
[ODB] Odebrane dane:     1111
```

*Rysunek 13*
*Ciągi binarne dla częstych błędów*

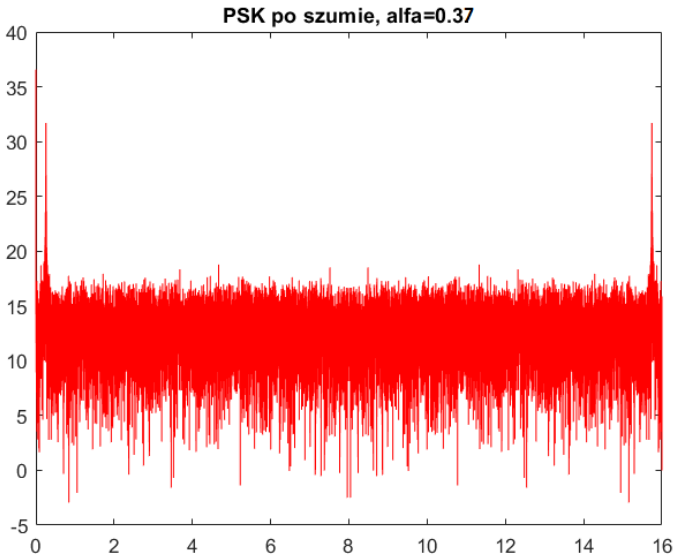



*Rysunek 14*
*Wykres dla częstych błędów*

```
Napis po kodowaniu: 1100110011010010
Sygnal po demodula: 1111111111111111

[ODB] Odebrana wiadomosc:11111111
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:000
[ODB] Dane poprawne!
[ODB] Odebrane dane:     1111


[ODB] Odebrana wiadomosc:11111111
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:000
[ODB] Dane poprawne!
[ODB] Odebrane dane:     1111

Odebrany napis: 11111111
Odebrana wiadomosc:
```

*Rysunek 15*
*Ciągi binarne dla bardzo częstych błędów*

Tutaj można zaobserwować sytuację, że sam Hamming nie wystarczy do dekodowania. Odebrano poprawnie sformułowaną wiadomość (konsola tylko nie obsłużyła znaku), jednak nie jest to wysłana przez nas wiadomość.



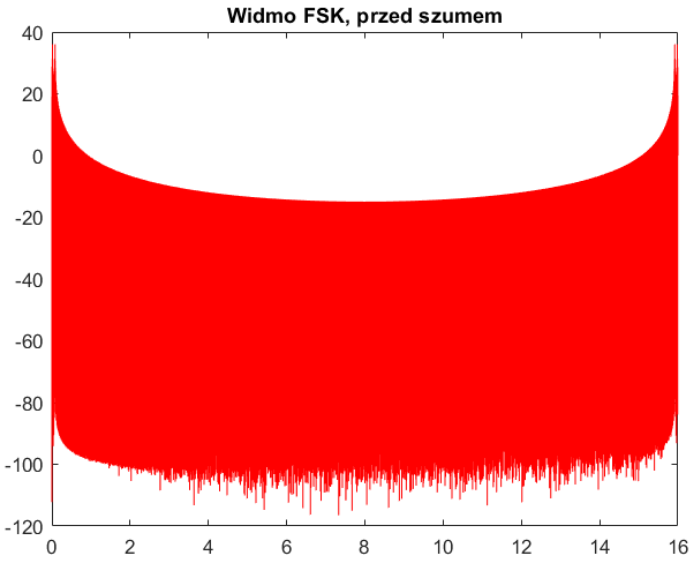

*Rysunek 16*
*Wykres dla bardzo częstych błędów*

## FSK

Dla FSK zostały znalezione 3 wartości alfa:

- Sporadyczne błędy: 0.13
- Częste błędy: 0.05
- Bardzo częste błędy: 0.001

Widmo bez szumu prezentuje się następująco:

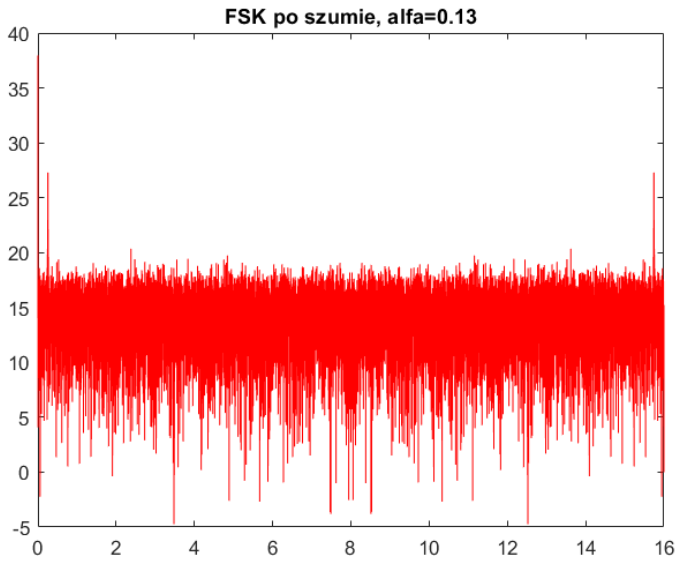



*Rysunek 17*
*Wykres widma dla FSK przed szumem*

Poniżej wykresy oraz ciągi binarne dla kolejnych wartości alfa:

```
Napis: a
Ciag binarny: 01100001
[NAD] Dane do wyslania:  0110
[NAD] Nadana wiadomosc:  11001100

[NAD] Dane do wyslania:  0001
[NAD] Nadana wiadomosc:  11010010

Napis po kodowaniu: 1100110011010010
Sygnal po demodula: 1110110011010010

[ODB] Odebrana wiadomosc:11101100
[ODB] Bit parzystosci:   1
[ODB] Odkodowana macierz:011
[ODB] Pojedynczy blad!
[ODB] Blad na pozycji:   2
[ODB] Blad naprawiony!
[ODB] Odebrane dane:     0110


[ODB] Odebrana wiadomosc:11010010
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:000
[ODB] Dane poprawne!
[ODB] Odebrane dane:     0001

Odebrany napis: 01100001
Odebrana wiadomosc: a
```

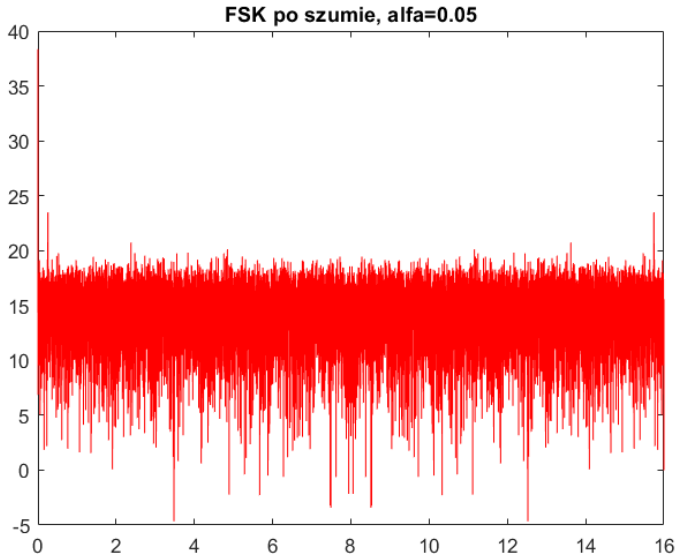
*Rysunek 18*
*Ciągi binarne dla sporadycznych błędów*




*Rysunek 19*
*Wykres dla sporadycznych błędów*

```
Napis po kodowaniu: 1100110011010010
Sygnal po demodula: 1010100110110011

[ODB] Odebrana wiadomosc:10101001
[ODB] Bit parzystosci:   0
[ODB] Odkodowana macierz:111
[ODB] Podwojny blad!
[ODB] Pakiet odrzucony!


[ODB] Odebrana wiadomosc:10110011
[ODB] Bit parzystosci:   1
[ODB] Odkodowana macierz:001
[ODB] Pojedynczy blad!
[ODB] Blad na pozycji:   0
[ODB] Blad naprawiony!
[ODB] Odebrane dane:     1001
```

*Rysunek 20*
*Ciągi binarne dla częstych błędów*

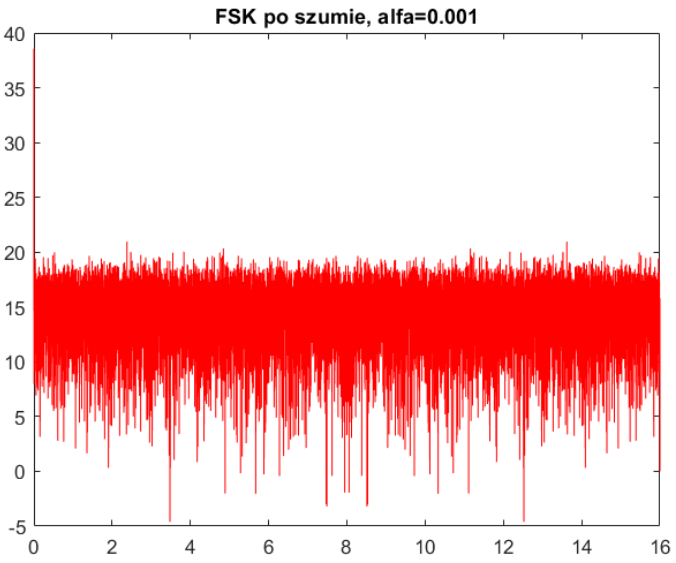



*Rysunek 21*
*Wykres dla częstych błędów*

```
Napis po kodowaniu: 1100110011010010
Sygnal po demodula: 1011100110110011

[ODB] Odebrana wiadomosc:10111001
[ODB] Bit parzystosci:   1
[ODB] Odkodowana macierz:011
[ODB] Pojedynczy blad!
[ODB] Blad na pozycji:   2
[ODB] Blad naprawiony!
[ODB] Odebrane dane:     0100


[ODB] Odebrana wiadomosc:10110011
[ODB] Bit parzystosci:   1
[ODB] Odkodowana macierz:001
[ODB] Pojedynczy blad!
[ODB] Blad na pozycji:   0
[ODB] Blad naprawiony!
[ODB] Odebrane dane:     1001

Odebrany napis: 01001001
Odebrana wiadomosc: I
```

*Rysunek 22*
*Ciągi binarne dla bardzo częstych błędów*

Podobna sytuacja jak w poprzednim przypadku, zostało zamienione tyle bitów i w taki sposób, że Hamming nie wykrył tego jako błąd i odebrał inną wiadomość niż nadana.




*Rysunek 23*
*Wykres dla bardzo częstych błędów*