Dominik Ciesiołkiewicz 44289 – Sprawozdanie TD 9. Budowa Toru Transmisyjnego.

Niestety pomimo usilnych prób nie udało mi się wykonać tego zadania. Stworzony przeze mnie kod produkuje prawidłową informację, dzieli ją na pakiety i koduje kodem Hamminga, a następnie używa modulacji ASK, FSK bądź PSK. Wygenerowany wykres do ASK załączam niżej. Problem pojawia się jednak przy demodulacji, gdyż wyniki zwracane przez algorytm są nieprawidłowe. Bardzo bym prosił o zerknięcie w mój kod i o wskazówkę, gdzie może znajdować się błąd. Pracowałem nad tym kodem bardzo długo, lecz nie jestem w stanie znaleźć błędu. Dane wyjściowe kodu znajdują się w plikach na repozytorium, kolejno:

- Dane.txt – zapis binarny stringa wpisanego do przetworzenia,
- Hamming.txt – dane dobrane w pakiety i zakodowane kodem Hamminga,
- ASK.txt – dane zmodulowane modulacją ASK,
- Zdemodulowane.txt – dane po demodulacji,
- DecodedReduced.txt – dane po zdekodowaniu; powinny wyglądać jak dane wejściowe.

Sądzę, że błąd może wynikać z tego, że używam pewnego rodzaju rozszerzenia traktując podczas modulacji. Traktuję wtedy każdy bit jako 8 (np. 01 traktuję jako 0000000011111111) i niepoprawnie demoduluję tę wiadomość, ale nie rozumiem czemu mój kod z zajęć z modulacji ASK, FSK i PSK działał wtedy poprawnie. Bardzo dziękuję za wszelkie wskazówki. Poprawiony kod postaram się wysłać do następnych zajęć.

Kod:

```
#include <iostream>
#include <fstream>
#include <complex>

using namespace std;

double pi = 3.14159265359;

int lengthOfString(string str)
{
   return str.length();
}

string S2BS(string in, bool choice)  //String To Binary Stream
{
   string out = "";

   int n = in.length();
   string bity = "";

   if (choice == 1)//LittleEndian
   {
        for (int i = 0; i < n; i++)
        {
            int wartosc = int(in[i]);
            bity = "";

            while (wartosc > 0)
            {
                if (wartosc % 2)
                {
                    bity += '1';
                }
                else
                {
                    bity += '0';
                }
                wartosc = wartosc / 2;
            }
            out += bity;
        }
        reverse(out.begin(), out.end());
        //cout << out << endl;
        return out;
    }
    else {//BigEndian
        for (int i = 0; i < n; i++)
        {
            int wartosc = int(in[i]);
            bity = "";

            while (wartosc > 0)
            {
                if (wartosc % 2)
                {
                    bity += '1';
                }
                else
                {
                    bity += '0';
                }
                wartosc = wartosc / 2;
            }
            reverse(bity.begin(), bity.end());
            out += bity;
        }
        //cout << out << endl;
        return out;
    }
}
```

```
int* Hamming(string d)
{
    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* K = new int[7];

    for (int i = 0; i < 7; i++)
    {
        K[i] = 0;
        for (int j = 0; j < 4; j++)
        {
            //cout << G[i][j];
            //cout<<d[j]<<endl;
            K[i] += G[i][j] * (d[j] - '0');
        }
        //cout << endl;
    }

    /*cout << "K:" << endl;
    for (int i = 0; i < 7; i++)
    {
        cout << K[i] << endl;
    }
    cout << endl;

    cout << "K modulo 2:" << endl;*/
    for (int i = 0; i < 7; i++)
    {
        K[i] = K[i] % 2;
        //cout << K[i] << endl;
    }
    //cout << endl;

    return K;
}

int* HammingSECDEC(string d)
{
    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* K = new int[8];

    for (int i = 0; i < 7; i++)
    {
        K[i] = 0;
        for (int j = 0; j < 4; j++)
        {
            K[i] += G[i][j] * (d[j] - '0');
        }
    }

    /*cout << "K:" << endl;
    for (int i = 0; i < 7; i++)
    {
        cout << K[i] << endl;
    }
    cout << endl;

    cout << "K modulo 2:" << endl;*/
    for (int i = 0; i < 7; i++)
    {
        K[i] = K[i] % 2;
        //cout << K[i] << endl;
    }
    //cout << endl;

    //Dla SECDEC:
    //cout << "Ze sprawdzajacym bitem: " << endl;
    int err = 0;
    for (int i = 0; i < 7; i++)
    {
        err += K[i];
    }
    err = err % 2;
    K[7] = err;

    /*for (int i = 0; i < 8; i++)
    {
        cout << K[i] << endl;
    }
    cout << endl;*/

    return K;
}

int* DecHamming(int* K)
{
    int H[3][7] = { {1,0,1,0,1,0,1},{0,1,1,0,0,1,1},{0,0,0,1,1,1,1} };
    int* KD = new int[7];

    for (int i = 0; i < 3; i++)
    {
        KD[i] = 0;
        for (int j = 0; j < 7; j++)
        {
            KD[i] += H[i][j] * K[j];
        }
    }

    for (int i = 0; i < 3; i++)
    {
        KD[i] = KD[i] % 2;
    }

    return KD;
}

int* DecHammingSECDEC(int* K)
{
    int H[3][7] = { {1,0,1,0,1,0,1},{0,1,1,0,0,1,1},{0,0,0,1,1,1,1} };
    int* KD = new int[7];

    cout << "Sprawdzanie p4:" << endl;
    int err = 0;
    for (int i = 0; i < 7; i++)
    {
        err += K[i];
    }
    err = err % 2;

    if (err != K[7])
    {
        cout << "P4 nie jest zgodne. Mamy 50% szans na powodzenie naprawy." << endl << endl;
    }
    else
    {
        cout << "P4 jest zgodne" << endl << endl;
    }

    int p1 = (K[0] + K[2] + K[4] + K[6]) % 2;
    int p2 = (K[1] + K[2] + K[5] + K[6]) % 2;
    int p3 = (K[3] + K[4] + K[5] + K[6]) % 2;

    int n = p1 * 1 + p2 * 2 + p3 * 4 - 1;

    cout << "Poprawiony kod odebrany:" << endl;
    if (K[n] == 0)
    {
        K[n] = 1;
    }
    else
    {
        K[n] = 0;
    }
    for (int i = 0; i < 8; i++)
    {
        cout << K[i] << endl;
    }

    cout << endl << "Sprawdzanie p4 - ponowne:" << endl;
    n = 0;
    for (int i = 0; i < 7; i++)
    {
        n += K[i];
    }
    n = n % 2;

    if (n != K[7])
    {
        cout << "P4 nie jest zgodne. Sa co najmniej 2 bledne bity. Odrzucamy pakiet." << endl << endl;
        return NULL;
    }
    else
    {
        cout << "P4 jest zgodne, odkodowujemy:" << endl << endl;

        cout << "Informacja odkodowana:" << endl;
        cout << K[2] << endl;
        cout << K[4] << endl;
        cout << K[5] << endl;
        cout << K[6] << endl;
    }

    return K;
}

int* BitNegation(int* K, int NoBit)
{
    if (K[NoBit] == 0)
        K[NoBit] = 1;
    else
        K[NoBit] = 0;
    return K;
}

int* Mgenerator(string tab, int size, double Tb, double fs)
{
    ofstream saveM("M.txt");
    int probki = fs * Tb;
    int* m = new int[size * probki * 8];
    int index = 0;
    /*for (int i = 0; i < size/8; i++)
    {
        for (int j = 7; j >= 0; j--)
        {
            for (int k = 0; k < probki; k++)
            {
                if (tab[i]=='1' & (1 << j))
                {
                    m[index] = 1;
                }
                else
                {
                    m[index] = 0;
                }
                saveM << m[index] << endl;
                index++;
            }
        }
    }
    */
    for (int i = 0; i < size; i++)
    {
        if (tab[i] == '1')
        {
            for (int j = 0; j < 8 * probki; j++)
            {
                m[index] = 1;
                saveM << m[index] << endl;
                index++;
            }
        }
        else
        {
            for (int j = 0; j < 8 * probki; j++)
            {
                m[index] = 0;
                saveM << m[index] << endl;
                index++;
            }
        }
    }

    saveM.close();
    return m;
}

int* MgeneratorSTR(int* tab, int size, double Tb, double fs)
{
    ofstream saveM("M.txt");
    int probki = fs * Tb;
    int* m = new int[size * probki * 8];
    int index = 0;
    /*for (int i = 0; i < size/8; i++)
    {
        for (int j = 7; j >= 0; j--)
        {
            for (int k = 0; k < probki; k++)
            {
                if (tab[i]=='1' & (1 << j))
                {
                    m[index] = 1;
                }
                else
                {
                    m[index] = 0;
                }
                saveM << m[index] << endl;
                index++;
            }
        }
    }
    */
    for (int i = 0; i < size; i++)
    {
        if (tab[i] == 1)
        {
            for (int j = 0; j < 8 * probki; j++)
            {
                m[index] = 1;
                saveM << m[index] << endl;
                index++;
            }
        }
        else
        {
            for (int j = 0; j < 8 * probki; j++)
            {
                m[index] = 0;
                saveM << m[index] << endl;
                index++;
            }
        }
    }

    saveM.close();
    return m;
}
```

```
int* MgeneratorSTRv2(int* tab, int size, double Tb, double fs)
{
    ofstream saveM("M.txt");
    int probki = fs * Tb;
    int* m = new int[size * probki];
    int index = 0;

    for (int i = 0; i < size; i++)
    {
        if (tab[i] == 1)
        {
            m[index] = 1;
            saveM << m[index] << endl;
            index++;
        }
        else
        {
            m[index] = 0;
            saveM << m[index] << endl;
            index++;
        }
    }

    saveM.close();
    return m;
}

int* clock(double f, int size, double Tb, double fs)
{
    ofstream saveClock("Clock.txt");
    int probki = fs * Tb;
    int* clock = new int[size * probki * 8];

    double phase = 0;
    for (int i = 0; i < size * probki * 8; i++) {

        if (phase < 0.5)
        {
            clock[i] = 1;
        }
        else
        {
            clock[i] = 0;
        }

        phase += f / (probki * 8);
        if (phase >= 1)
        {
            phase -= 1;
        }
        saveClock << clock[i] << endl;
    }
    saveClock.close();
    return clock;
}

double* timeSpan(double f, int size, double Tb, double fs)
{
    ofstream saveTimeSpan("Time.txt");
    int probki = fs * Tb;
    double* time = new double[size * probki * 8];

    double timeStamp = 0;

    for (int i = 0; i < size * probki * 8; i++) {
        //timeStamp = double(double(i) / double(fs));
        time[i] = timeStamp;
        saveTimeSpan << time[i] << endl;
        timeStamp += double(1 / (double(probki) * 8));
    }

    saveTimeSpan.close();
    return time;
}

double* TTLCoder(int size, double Tb, double fs, int* m, int* clock)
{
    ofstream saveTTL("TTL.txt");
    int probki = fs * Tb;
    double* TTL = new double[size * probki * 8];
    TTL[0] = m[0];
    saveTTL << TTL[0] << endl;
    for (int i = 1; i < size * probki * 8; i++)
    {
        if (clock[i] == 1 && clock[i] != clock[i - 1])
        {
            if (m[i] == 1)
            {
                TTL[i] = 1;
            }
            else
            {
                TTL[i] = 0;
            }
        }
        else
        {
            TTL[i] = TTL[i - 1];
        }
        saveTTL << TTL[i] << endl;
    }

    saveTTL.close();
    return TTL;
}

int* TTLDecoder(int size, double Tb, double fs, double* m, int* clock)
{
    ofstream saveDecTTL("DecTTL.txt");
    int probki = fs * Tb;
    int* decoded = new int[size * probki * 8];
    decoded[0] = 1;
    saveDecTTL << decoded[0] << endl;
    for (int i = 1; i < size * probki * 8; i++)
    {
        if (clock[i] == 0 && clock[i] != clock[i - 1])
        {
            decoded[i] = m[i];
        }
        else
        {
            decoded[i] = decoded[i - 1];
        }
        saveDecTTL << decoded[i] << endl;
    }

    saveDecTTL.close();
    return decoded;
}

complex<double>* DFT(const double* tab, int N)
{
    complex<double>* tab2 = new complex<double>[N];

    for (int k = 0; k < N; k++)
    {
        tab2[k] = 0;
        complex<double> WN = cos(tab[k]) + 1i * sin(tab[k]);

        for (int n = 0; n < N; n++)
        {
            tab2[k] += tab[n] * pow(WN, -k * n);
        }

        //for (int n = 0; n < N; n++)
        //{
        //    tab2[k] += tab[n] * exp(-2 * pi * 1i * (double)k * (double)n / (double)N);
        //}

    }

    return tab2;
}

double ton_prosty(double A1, double F, double t)// czy jest w ogóle potrzebny?
{
    return A1 * sin(2 * pi * F * t);
}

double* ASK(int* m, int n, int A1, int A2, double f, double fs, double phi)
{
    double* zA = new double[n];
    for (int i = 0; i < n; i++)
    {
        if (m[i] == 0)
        {
            zA[i] = A1 * sin(2 * pi * f * i / fs + phi);
        }
        else
        {
            zA[i] = A2 * sin(2 * pi * f * i / fs + phi);
        }
    }
    /*ofstream saveASK("ASK.txt");
    for (int i = 0; i < n; i++)
    {
        saveASK << zA[i] << endl;
    }
    saveASK.close();*/
    return zA;
}

double* FSK(int* m, int n, int A, int N, double fs, double Tb, double phi)
{
    double* zF = new double[n];
    double f0 = (N + 1) / Tb;
    double f1 = (N + 2) / Tb;
    for (int i = 0; i < n; i++)
    {
        if (m[i] == 0)
        {
            zF[i] = A * sin(2 * pi * f0 * i / fs + phi);
        }
        else
        {
            zF[i] = A * sin(2 * pi * f1 * i / fs + phi);
        }
    }
    ofstream saveFSK("FSK.txt");
    for (int i = 0; i < n; i++)
    {
        saveFSK << zF[i] << endl;
    }
    saveFSK.close();
    return zF;
}

double* PSK(int* m, int n, int A, double f, double fs, double Tb)
{
    double* zP = new double[n];
    for (int i = 0; i < n; i++)
    {
        if (m[i] == 0)
        {
            zP[i] = A * sin(2 * pi * f * i / fs + 0);
        }
        else
        {
            zP[i] = A * sin(2 * pi * f * i / fs + pi);
        }
    }
    ofstream savePSK("PSK.txt");
    for (int i = 0; i < n; i++)
    {
        savePSK << zP[i] << endl;
    }
    savePSK.close();
    return zP;
}

double* sinusoid(double f, double phi, double A, double fs, int probki)
{
    double* sinus = new double[probki];
    for (int i = 0; i < probki; i++) {
        sinus[i] = A * sin(2 * pi * i / fs * f + phi);
    }
    return sinus;
}
```

```
int* demodulatorASKPSK(double* pasmo, int n, double h, double fs, double f, double A)
{
  //Faza 1:

  double* Sinus = sinusoid(f, 0, A, fs, n);
  double* x = new double[n];
  for (int i = 0; i < n; i++) {
    x[i] = pasmo[i] * Sinus[i];
  }

  //Faza 2 i 3:
  double* pt = new double[n];
  int* mt = new int[n];

  double calka;
  for (int i = 0; i < n; i++)
  {
    double suma = 0;

    if (i % 625 == 0)
      calka = 0;
    calka += x[i];

    if (calka >= h)
    {
      mt[i] = 1;
    }
    else
    {
      mt[i] = 0;
    }
  }

  return mt;
}

int* demodulatorFSK(double* pasmo, int n, double h, double fs, double f1, double f2, double A)
{
  //Faza 1:
  double* x1 = new double[n];
  double* x2 = new double[n];
  double calka1;
  double calka2;
  double* Sinus1 = sinusoid(f1, 0, A, fs, n);
  double* Sinus2 = sinusoid(f2, 0, A, fs, n);

  for (int i = 0; i < n; i++) {
        x1[i] = pasmo[i] * Sinus1[i];
        x2[i] = pasmo[i] * Sinus2[i];
    }

    //Faza 2 i 3:
    double* pt = new double[n];
    int probkiNaBit = 2;
    int* mt = new int[n];

    double p;
    for (int i = 0; i < n; i++)
    {
        double suma = 0;

        if (i % 625 == 0)
        {
            calka1 = 0;
            calka2 = 0;
        }
        calka1 += x1[i];
        calka2 += x2[i];

        p = calka2 - calka1;

        if (p >= h)
        {
            mt[i] = 1;
        }
        else
        {
            mt[i] = 0;
        }
    }

    return mt;
}

int main()
{
    double Tb = 0.1; //[s]
    int fs = 10000; //[Hz]

    //ASK:
    double A1 = 1.0;
    double A0 = 0.0;
    int f = 100;
    //FSK:
    double A = 1.0;
```

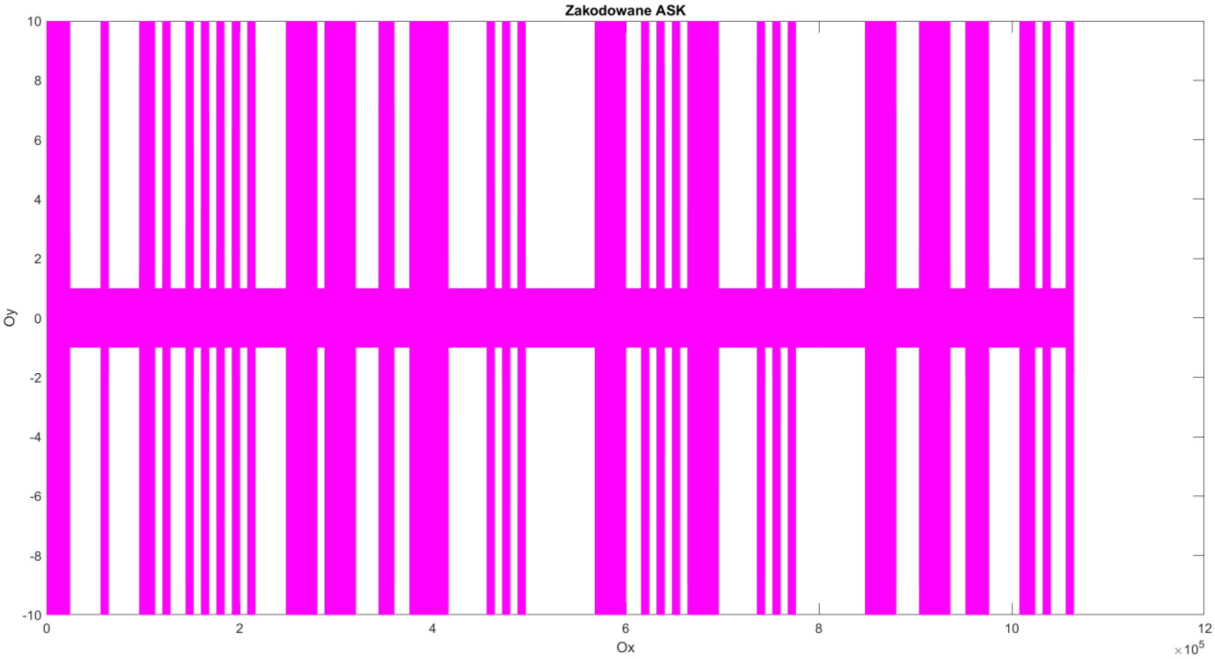
```
int f1 = 125;
int f0 = 250;
//PSK:
double phi0 = 0.0;
double phi1 = 180.0;//[rad]

//WCZYTYWANIE INFORMACJI
string str = S2BS("ALA MA KOTA", 1);
int n = lengthOfString(str);
cout << "Ilosc bitow transmisji: " << n << endl;

ofstream saveData("Dane.txt");

cout << "Informacja:" << endl;
for (int i = 0; i < n; i++)
{
    cout << str[i];
}
cout << endl << endl;
saveData << str << endl;

saveData.close();

bool SECDEC = 0;// 0-zwykly kod Hamminga; 1-SECDEC
int MOD = 0;//0-ASK, 1-FSK, 2-PSK


//PAKIETOWANIE
double nrOfTran = (double)n / 4;
int completeNrOfTran = (int)nrOfTran;
//cout << completeNrOfTran << endl;

int count = 0;
string dane[50];
int diff = 0;
for (int i = 0; i < n; i += 4)
{
    dane[count] = str.substr(i, 4);
    count++;
    diff = i;
}

//for (int i = 0; i < completeNrOfTran; i++)
//    cout << dane[i];
//cout << endl;

int reszta = n - (completeNrOfTran * 4);
//cout << reszta << endl;
```

```
dane[count - 1] = str.substr(diff, 4);
//cout << dane[count] << endl;

diff = 4 - (dane[count - 1].length());
//cout << diff << endl;

for (int i = 0; i < diff; i++)
    dane[count - 1].insert(0, "0");
//cout << dane[count-1] << endl;

//cout << count << endl;
//cout << completeNrOfTran + 1 << endl;

//KODOWANIE KODEM HAMMINGA
int** K = new int* [completeNrOfTran + 1];
for (int i = 0; i < completeNrOfTran + 1; i++)
    K[i] = new int[7];

if (SECDEC == 0)
{
    for (int i = 0; i < count; i++)
    {
        //cout << dane[i] << endl;
        K[i] = Hamming(dane[i]);
    }

    ofstream kodHamming("Hamming.txt");

    cout << "Informacja zakodowana kodem Hamminga:" << endl;
    for (int i = 0; i < count; i++)
    {
        for (int j = 0; j < 7; j++)
        {
            cout << K[i][j];
            kodHamming << K[i][j];
        }
        kodHamming << endl;
        cout << endl;
    }
    //K = BitNegation(K, 2);

    kodHamming.close();

    //cout << "K po negacji bitu 2:" << endl;
    //for (int i = 0; i < 7; i++)
    //{
    //    cout << K[i] << endl;
    //}
    //cout << endl;
}
else
{
    int* K = HammingSECDEC(str);

    K = BitNegation(K, 2);

    //cout << "K po negacji bitu 2:" << endl;
    //for (int i = 0; i < 8; i++)
    //{
    //    cout << K[i] << endl;
    //}
    //cout << endl;
}

//MODULACJA

int A2 = 10;
int N = 2;
int probki = fs * Tb;
double phi = 0;
int msize = 7 * probki * 8;

if (MOD == 0)
{
    ofstream saveASK("ASK.txt");
    for (int i = 0; i < completeNrOfTran + 1; i++)
    {
        int* m = MgeneratorSTR(K[i], n, Tb, fs);
        double* asktab = ASK(m, msize, A1, A2, f, fs, phi);
        for (int i = 0; i < msize; i++)
        {
            saveASK << asktab[i] << endl;
        }
    }
    saveASK.close();
}
else if (MOD == 1)
{
    ofstream saveFSK("FSK.txt");
    for (int i = 0; i < completeNrOfTran + 1; i++)
    {
        int* m = MgeneratorSTR(K[i], n, Tb, fs);
        double* fsktab = FSK(m, msize, A, N, fs, Tb, phi);
        for (int i = 0; i < msize; i++)
        {
            saveFSK << fsktab[i] << endl;
        }
    }
    saveFSK.close();
}
else
{
    ofstream savePSK("PSK.txt");
    for (int i = 0; i < completeNrOfTran + 1; i++)
    {
        int* m = MgeneratorSTR(K[i], n, Tb, fs);
        double* psktab = PSK(m, msize, A, f, fs, Tb);
        for (int i = 0; i < msize; i++)
        {
            savePSK << psktab[i] << endl;
        }
    }
    savePSK.close();
}

//DEMODULACJA

ifstream inFile;

if (MOD == 0)
{
    inFile.open("ASK.txt");
}
else if (MOD == 1)
{
    inFile.open("FSK.txt");
}
else
{
    inFile.open("PSK.txt");
}

if (!inFile) {
    cerr << "Nie odnaleziono pliku z danymi do demodulacji." << endl;
    exit(1);
}

double* modarr = new double[(completeNrOfTran + 1)*msize];
double x;
int size = 0;

//ofstream odczytane("Odczytane.txt");
while (inFile >> x) {
    modarr[size] = x;
    //odczytane << x;
    size++;
}
//odczytane.close();
inFile.close();

int* dem;

if (MOD == 0)
{
    dem = demodulatorASKPSK(modarr, msize, 400, fs, f, A);
}
else if (MOD == 1)
{
    dem = demodulatorASKPSK(modarr, msize, 0, fs, f, A);
}
else
{
    dem = demodulatorFSK(modarr, msize, 0, fs, f0, f1, A);
}


ofstream demodulator("Zdemodulowane.txt");
int counterdem = 0;
for (int i = 0; i < msize; i++)
{
    demodulator << dem[i];
    counterdem++;
    if (counterdem = 8)
    {
        counterdem = 0;
        demodulator << endl;
    }
}

demodulator.close();

//PONOWNE PAKIETOWANIE
int** D;

size = 0;
int counter = 0;

if (SECDEC == 0)
{
    D = new int* [msize / 7];
    for (int i = 0; i < msize / 7 + 1; i++)
```

```
        D[i] = new int[7];

    for (int i = 0; i < msize; i++)
    {
        D[size][counter] = dem[i];
        counter++;
        if (counter == 7)
        {
            counter = 0;
            size++;
        }
    }
}
else
{
    D = new int* [msize / 8];
    for (int i = 0; i < msize / 8 + 1; i++)
        D[i] = new int[7];

    for (int i = 0; i < msize; i++)
    {
        D[size][counter] = dem[i];
        counter++;
        if (counter == 7)
        {
            counter = 0;
            size++;
        }
    }
}

/*for (int i = 0; i < msize / 7; i++)
    for (int j = 0; j < 7; j++)
        cout << D[i][j] << endl;*/

//DEKODOWANIE

int* decodedInfo;

if (SECDEC == 0)
{
    decodedInfo = new int[msize / 7];
    ofstream decoded("Decoded.txt");

    for (int it = 0; it < msize / 7; it++)
    {
        int* TD = DecHamming(D[it]);
        if (TD[0] == 0 && TD[1] == 0 && TD[2] == 0)
        {
            //cout << "Kod nie posiada bledu. Transmisja poprawna." << endl << endl;
        }
        else
        {

            int err = (TD[0] + TD[1] * 2 + TD[2] * 4) - 1;

            if (D[it][err] == 0)
            {
                D[it][err] = 1;
            }
            else
            {
                D[it][err] = 0;
            }

            /*for (int i = 0; i < 7; i++)
            {
                cout << K[i] << endl;
            }
            cout << endl;*/
        }

        decoded << D[it][2] << D[it][4] << D[it][5] << D[it][6] << endl;
    }
    decoded.close();

    ofstream decodedRed("DecodedReduced.txt");
    count = 600;
    for (int i = 0; i < msize / 7; i++)
    {
        if (count == 600)
        {
            decodedRed << D[i][2] << D[i][4] << D[i][5] << D[i][6];
            count = 0;
        }
        count++;
    }
    decodedRed.close();
}
else
{
    decodedInfo = new int[msize / 8];

    //int* D = DecHammingSECDEC(K);
}
```

```
return 1;
}
```

Wykres:


Zakodowane ASK
Oy
Ox
×10^5