Transmisja Danych – Lab 09

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>

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++)
        {
```

```
64                  //cout << C[j][0] << "+(" << D[0][i]
65                  C[j][0] = C[j][0] + (D[i][0] * G[j][i]);
66              }
67              C[j][0] = C[j][0] % 2;
68              BitKontrolny = BitKontrolny + C[j][0];
69          }
70
71          C[7][0] = BitKontrolny % 2;
72
73          return C;
74      }
75
76      string Hamming_Nadajnik(string S)
77      {
78          cout << "[NAD] Dane do wyslania:  " << S << endl;
79          int** C = new int* [4];
80
81          for (int j = 0; j < 4; j++)
82          {
83              C[j] = new int[1];
84          }
85
86          for (int j = 0; j < 4; j++)
87          {
88              C[j][0] = (int)S[j] - '0'; // Konwersja na liczbę
89          }
90
91          int** Result = G_Multiply(C);
92          string X = "00000000";
93
94          for (int j = 0; j < 8; j++)
95          {
96              X[j] = Result[j][0] + 48; // Konwersja na znak
97          }
98          cout << "[NAD] Nadana wiadomosc:  " << X << endl;
99          return X;
100     }
101
102     void Zegar(double Czestotliwosc, double Probkowanie)
103     {
104         fstream File1;
105         File1.open("C:/Users/Qrystian/Desktop/results1.txt", ios::out);
106
107         int Check = 0;
108         int Bit = 1;
109         for (double j = 0; j < Czestotliwosc; j = j + Probkowanie)
110         {
111             File1 << j << " " << Bit << endl;
112
113             if (Check == 500)
114             {
115                 Check = 0;
116                 if (Bit == 0)
117                     Bit = 1;
118                 else
119                     Bit = 0;
120             }
121
122             Check++;
123         }
124
125         File1.close();
126     }
```

```
127
128 double TTL(char t)
129 {
130     double tt = (double)t - '0'; // Konwersja na liczbę
131
132     if (tt == 0)
133     {
134         return 0.0;
135     }
136
137     if (tt == 1)
138     {
139         return 1.0;
140     }
141 }
142
143 double zA(double A1, double A2, double f, double Fi, char T, double t)
144 {
145     double tt = (double)T - '0'; // Konwersja na liczbę
146     if (tt == 0)
147     {
148         return A1 * sin(2.0 * M_PI * f * t + Fi);
149     }
150
151     if (tt == 1)
152     {
153         return A2 * sin(2.0 * M_PI * f * t + Fi);
154     }
155 }
156
157 double zF(double A, long double f0, double f1, double Fi, char T, double t)
158 {
159     double tt = (double)T - '0'; // Konwersja na liczbę
160     if (tt == 0)
161     {
162         return A * sin(2.0 * M_PI * f0 * t + Fi);
163     }
164
165     if (tt == 1)
166     {
167         return A * sin(2.0 * M_PI * f1 * t + Fi);
168     }
169 }
170
171 double zP(double A, double f, double Fi1, double Fi2, char T, double t)
172 {
173     double tt = (double)T - '0'; // Konwersja na liczbę
174     if (tt == 0)
175     {
176         return A * sin(2.0 * M_PI * f * t + Fi1);
177     }
178
179     if (tt == 1)
180     {
181         return A * sin(2.0 * M_PI * f * t + Fi2);
182     }
183 }
184
185 string Hamming_Odbiornik(string S)
186 {
187     cout << endl;
188     cout << "[ODB] Odebrana wiadomosc:" << S << endl;
189     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} };
```

```
190     int** C = new int* [8];
191     int** Answer = new int* [3];
192     int BitParzystosci = 0;
193
194     for (int j = 0; j < 8; j++)
195     {
196         C[j] = new int[1];
197         C[j][0] = (int)S[j] - '0';
198         BitParzystosci = BitParzystosci + C[j][0];
199     }
200
201     BitParzystosci = BitParzystosci % 2;
202     cout << "[ODB] Bit parzystosci:   " << BitParzystosci << endl;
203
204     for (int j = 0; j < 3; j++)
205     {
206         Answer[j] = new int[1];
207         Answer[j][0] = 0;
208     }
209
210     cout << "[ODB] Odkodowana macierz:";
211
212     for (int j = 0; j < 3; j++)
213     {
214         for (int i = 0; i < 7; i++)
215         {
216             Answer[j][0] = Answer[j][0] + (C[i][0] * H[j][i]);
217         }
218         Answer[j][0] = Answer[j][0] % 2;
219         cout << Answer[j][0];
220     }
221     cout << endl;
222
223     bool Poprawnosc;
224     for (int j = 0; j < 3; j++)
225     {
226         if (Answer[j][0] == 0)
227         {
228             Poprawnosc = true;
229             continue;
230         }
231         else
232         {
233             Poprawnosc = false;
234             break;
235         }
236     }
237
238     if ((Poprawnosc == true) && (BitParzystosci == 0))
239     {
240         cout << "[ODB] Dane poprawne!" << endl;
241         string X = "0000";
242
243         X[0] = C[2][0] + 48;
244         X[1] = C[4][0] + 48;
245         X[2] = C[5][0] + 48;
246         X[3] = C[6][0] + 48;
247
248         cout << "[ODB] Odebrane dane:     " << X << endl << endl;
249         return X;
250     }
251
252     if ((Poprawnosc == false) && (BitParzystosci == 1))
```

```
253         {
254             cout << "[ODB] Pojedynczy blad!" << endl;
255             int Pozycja = 0;
256             for (int j = 0; j < 3; j++)
257             {
258                 Pozycja = Pozycja + (Answer[2 - j][0] * pow(2, j));
259             }
260
261             cout << "[ODB] Blad na pozycji:   " << Pozycja - 1 << endl;
262             if (C[Pozycja - 1][0] == 1)
263                 C[Pozycja - 1][0] = 0;
264             else
265                 C[Pozycja - 1][0] = 1;
266
267             cout << "[ODB] Blad naprawiony!" << endl;
268
269             string X = "0000";
270
271             X[0] = C[2][0] + 48;
272             X[1] = C[4][0] + 48;
273             X[2] = C[5][0] + 48;
274             X[3] = C[6][0] + 48;
275
276             cout << "[ODB] Odebrane dane:     " << X << endl << endl;
277             return X;
278         }
279
280         if ((Poprawnosc == false) && (BitParzystosci == 0))
281         {
282             cout << "[ODB] Podwojny blad!" << endl;
283             cout << "[ODB] Pakiet odrzucony!" << endl << endl;
284             return "Pakiet uszkodzony";
285         }
286     }
287
288 int main()
289 {
290     const char* Napis = "a";
291     string S = S2BS(Napis, "BigEndian");
292     cout << "Napis: " << Napis << endl << "Ciag binarny: " << S << endl;
293
294     string SSS = "000000000000000";
295
296     int Poz = 0;
297     for (int k = 0; k < S.length(); k++)
298     {
299         if (k % 4 == 0)
300         {
301             string SS = S.substr(Poz, 4);
302             SS = Hamming_Nadajnik(SS);
303             cout << endl;
304
305             for (int a = 0; a < 8; a++)
306             {
307                 if (Poz == 0)
308                     SSS[a] = SS[a];
309                 else
310                     SSS[a + 8] = SS[a];
311             }
312
313             Poz = Poz + 4;
314         }
315     }
```

```
316
317     cout << "Napis po kodowaniu: " << SSS << endl;
318     S = SSS;
319
320     fstream File2;
321     fstream File3;
322     File3.open("C:/Users/Qrystian/Desktop/results3.txt", ios::out);
323     File2.open("C:/Users/Qrystian/Desktop/results2.txt", ios::out);
324
325     Zegar(S.length(), 0.001);
326     double Suma = 0;
327     int Checker = 0;
328
329     string SS = "000000000000000";
330     int Sj = 0;
331
332     for (double t = 0; t < S.length(); t = t + 0.001)
333     {
334         File2 << t << " " << TTL(S[floor(t)]) << endl;
335
336         /*
337         // ASK
338         File3 << t << " " << zA(0.0, 1.0, ((double)S.length()/0.001) * pow(1000, -1), 2 * M_PI, S[floor(t)], t) << endl;
339         double Y = zA(0.0, 1.0, ((double)S.length() / 0.01) * pow(1000, -1), 2 * M_PI, S[floor(t)], t) * zA(1.0, 1.0, ((double)S.length() / 0.01) * pow(1000, -1), 2 * M_PI, S[floor(t)], t) * 0.01;
340
341         if (Checker == 999) // Co 1000 próbek = co jeden bit
342         {
343             for (double j = 0.0 + t; j < 1 + t; j = j + 0.01)
344             {
345                 if (Suma > 0.3)   // h=0.3, odzyskanie sygnału
346                 {
347                     SS[Sj] = '1';
348                     Sj++;
349                     break;
350                 }
351                 else
352                 {
353                     SS[Sj] = '0';
354                     Sj++;
355                     break;
356                 }
357             }
358             Suma = 0;
359             Checker = 0;
360         }
361         else
362         {
363             Suma = Suma + Y; // Sumowanie kolejnych próbek = drugi krok całkowania
364             Checker++;
365         }
366         */
367
368         // PSK
369         /*
370         File3 << t << " " << zP(1.0, ((double)S.length() / 0.01) * pow(1000, -1), 0.0, M_PI, S[floor(t)], t) << endl;
371         double Y = zP(1.0, ((double)S.length() / 0.01) * pow(1000, -1), 0.0, M_PI, S[floor(t)], t) * zP(1.0, ((double)S.length() / 0.01) * pow(1000, -1), M_PI, M_PI, S[floor(t)], t) * 0.01;
372
373         if (Checker == 999)
374         {
375             for (double j = 0.0 + t; j < 1 + t; j = j + 0.01)
376             {
377                 if (Suma > 0)   // h=0, odzyskanie sygnału
378                 {
```

```
                SS[Sj] = '1';
                Sj++;
                break;
            }
            else
            {
                SS[Sj] = '0';
                Sj++;
                break;
            }
        }
        Suma = 0;
        Checker = 0;
    }
    else
    {
        Suma = Suma + Y;
        Checker++;
    }
    */


    // FSK
    File3 << t << " " << zF(1.0, 1.0, 5.0, 2 * M_PI, S[floor(t)], t) << endl;
    double x1 = zF(1.0, 1.0, 5.0, 2 * M_PI, S[floor(t)], t) * zF(1.0, 1.0, 1.0, 2 * M_PI, S[floor(t)], t);
    double x2 = zF(1.0, 1.0, 5.0, 2 * M_PI, S[floor(t)], t) * zF(1.0, 5.0, 5.0, 2 * M_PI, S[floor(t)], t);
    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 demodulacji: " << SS << endl;

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:

```
Napis: a
Ciag binarny: 01100001
```

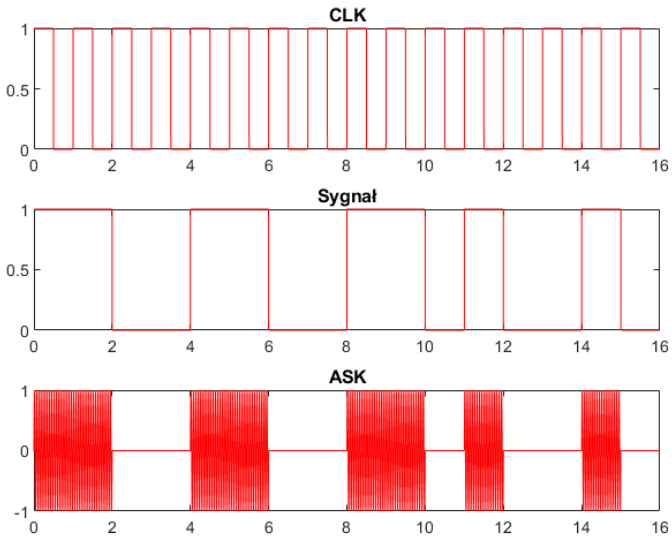
Rysunek 1
Napis i ciąg binarny

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

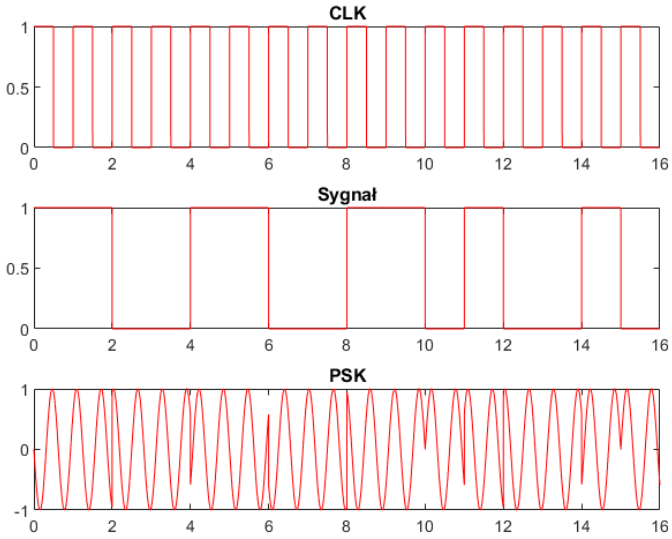
```
[NAD] Dane do wyslania:  0110
[NAD] Nadana wiadomosc:  11001100

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

Napis po kodowaniu: 1100110011010010
```

Rysunek 2
Zakodowany napis

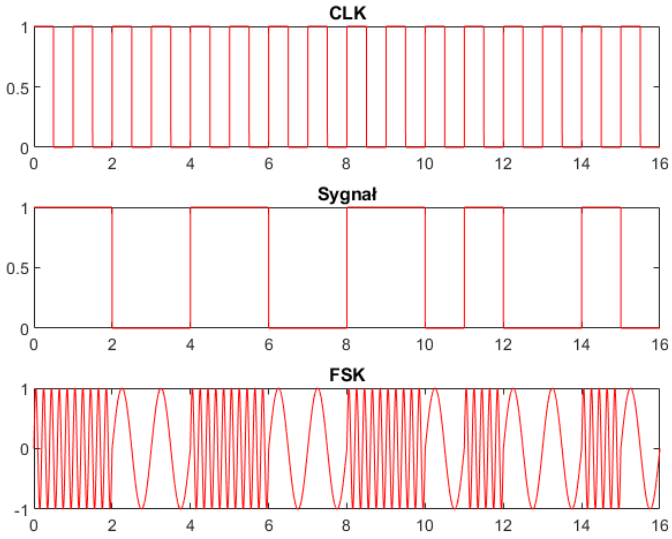
Następnie ten 16bitowy ciąg został kolejno zmodulowany ASK, FSK, PSK. Poniżej wykresy wszystkich:



Rysunek 3
Modulacja ASK



Rysunek 4
Modulacja PSK

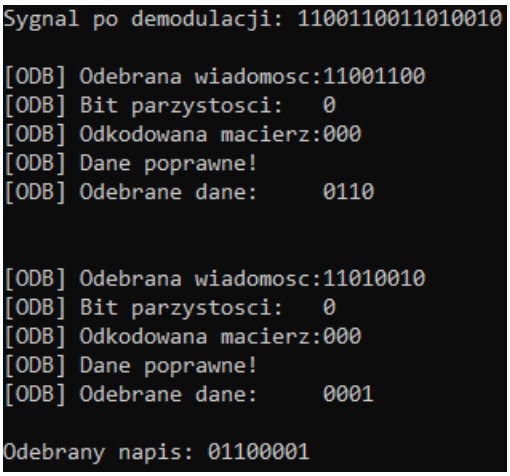


Rysunek 5
Modulacja FSK

W każdym przypadku, po demodulacji otrzymano następujący ciąg:

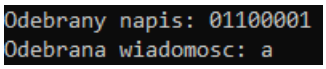
```
Sygnal po demodulacji: 1100110011010010
```

Następnie ten ciąg został poddany dekodowaniu kanałowemu, otrzymano następujący ciąg:



Rysunek 6
Dekodowanie w odbiorniku

Po przekształceniu ciągu, otrzymaliśmy naszą pierwotną wiadomość:



Rysunek 7
Odebrana wiadomość