Transmisja Danych – Lab 06

Krystian Bartosik 213A, nr 44266

Kod źródłowy:

```
// 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;
        }
    }
    // "test" = 01110100011100110110010101110100

    cout << result;
    return result;
}

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);
    }
}
```

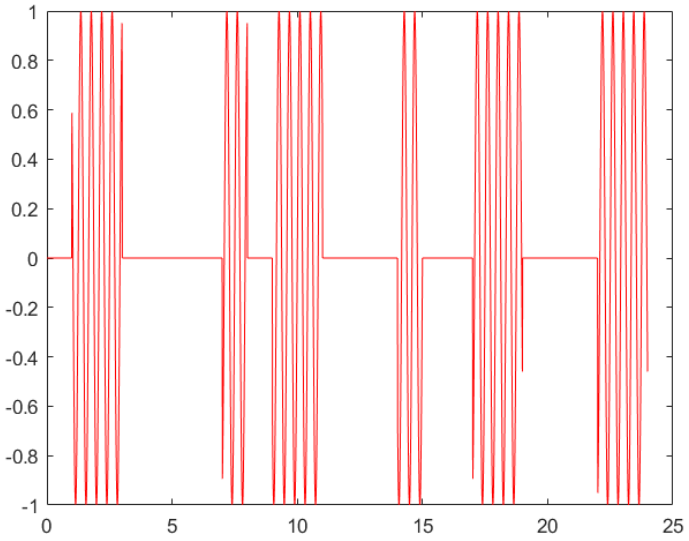
```
double zF(double A, long double f0, double f1, double Fi, char T, double t)
{
    double tt = (double)T - '0'; // Konwersja na liczbę
    if (tt == 0)
    {
        return A * sin(2.0 * M_PI * f0 * t + Fi);
    }

    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);
    }
}

int main()
{
    fstream File;
    File.open("C:/Users/Qrystian/Desktop/results.txt", ios::out);
    string S = S2BS("abc", "BigEndian"); // 24bity

    double Suma = 0;
    int Checker = 0;
    for (double t = 0; t < S.length(); t = t + 0.01) //zA(0.0, 1.0, ((double)S.length()/0.01) *
pow(1000, -1), 2 * M_PI, S[floor(t)], t)
    {
        // zA
        //File << t << " " << zA(0.0, 1.0, ((double)S.length()/0.01) * pow(1000, -1), 2 * M_PI,
S[floor(t)], t) << endl;
        //File << t << " " << 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) << endl;
        //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; // próbka * długość próbki = 1 krok całkowania

        /*
        if (Checker == 99) // Co 100 próbek = co jeden bit
        {
            for (double j = 0.0 + t; j < 1 + t; j = j + 0.01) // Wpisz 100 kolejnych próbek do
pliku
            {
                //File << j-1.0 << " " << Suma << endl; // Rysowanie funkcji po całkowaniu


                if (Suma > 0.3)   // h=0.3, odzyskanie sygnału
                    File << j << " " << 1 << endl;
                else
                    File << j << " " << 0 << endl;
            }
```

```
            Suma = 0;
            Checker = 0;
        }
        else{
            Suma = Suma + Y; // Sumowanie kolejnych próbek = drugi krok całkowania
            Checker++;}
        */

        // zP
        //File << t << " " << zP(1.0, ((double)S.length() / 0.01) * pow(1000, -1), 0.0, M_PI,
S[floor(t)], t) << endl;
        //File << t << " " << 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) << endl;
        //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;

        /*
        if (Checker == 99) // Co 100 próbek = co jeden bit
        {
            for (double j = 0.0+t; j < 1+t; j=j+0.01) // Wpisz 100 kolejnych próbek do pliku
            {//File << j-1.0 << " " << Suma << endl; // Rysowanie funkcji po całkowaniu
                if (Suma > 0)   // h=0, odzyskanie sygnału
                    File << j << " " << 1 << endl;
                else File << j << " " << 0 << endl;}
            Suma = 0;
            Checker = 0;
        }
        else{
            Suma = Suma + Y;
            Checker++;}
        */

        // zF
        //File << 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);
        //File << t << " " << x1 << endl;
        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);
        //File << t << " " << x2 << endl;
        double Y = (x2 * 0.01) - (x1 * 0.01);

        if (Checker == 99) // Co 100 próbek = co jeden bit
        {
            for (double j = 0.0 + t; j < 1 + t; j = j + 0.01)
            {
                //File << j-1.0 << " " << Suma << endl; // Rysowanie funkcji po całkowaniu
                if (Suma > 0)   // h=0, odzyskanie sygnału
                    File << j << " " << 1 << endl;
                else
                    File << j << " " << 0 << endl;
            }
            Suma = 0;
            Checker = 0;
        }
        else{
            Suma = Suma + Y;
            Checker++;
        }
    }

    File.close();
}
```

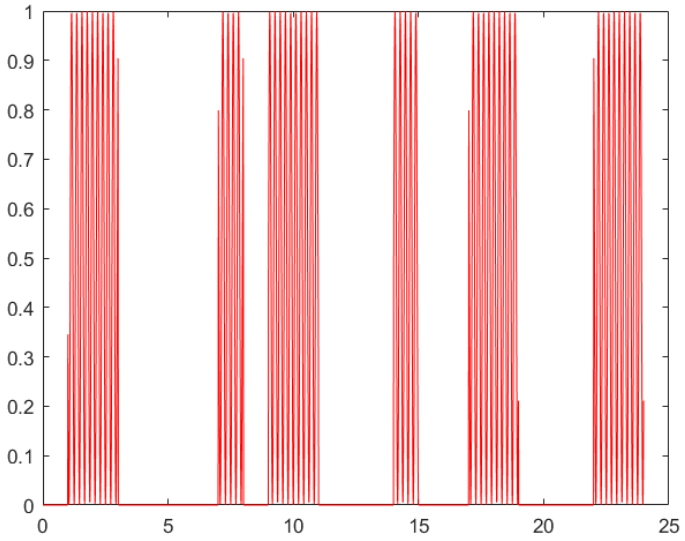
Opis kodu:

- Zadanie 1
- Zadanie 2
- Zadanie 3

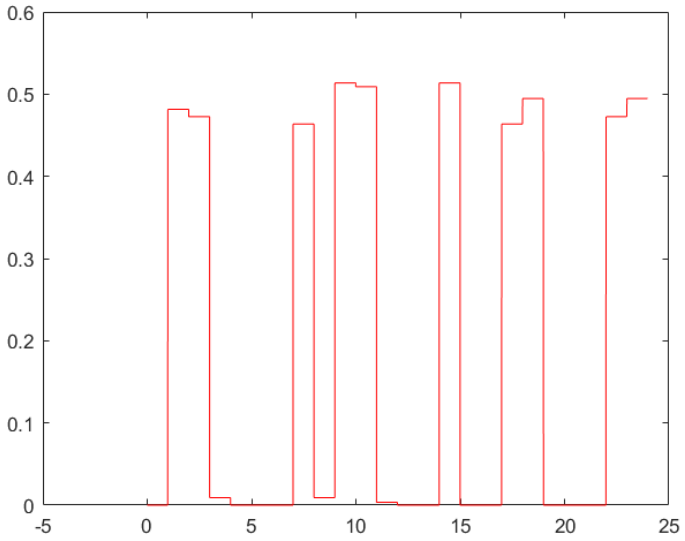
Wygenerowane wykresy:



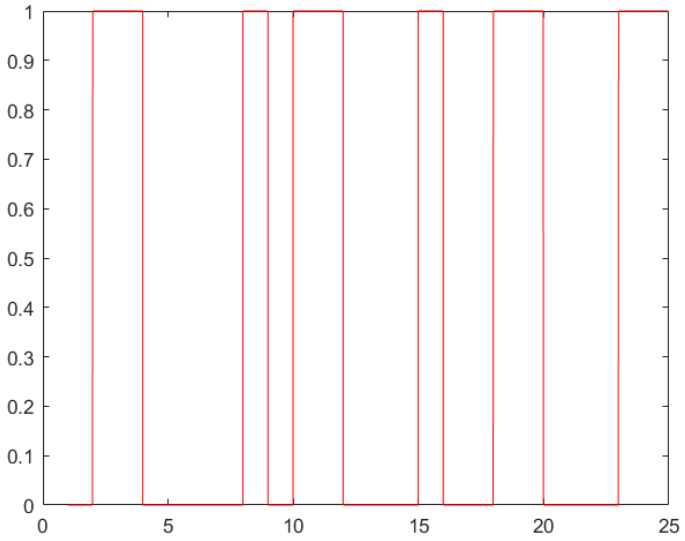
Wykres 1 zA(t)



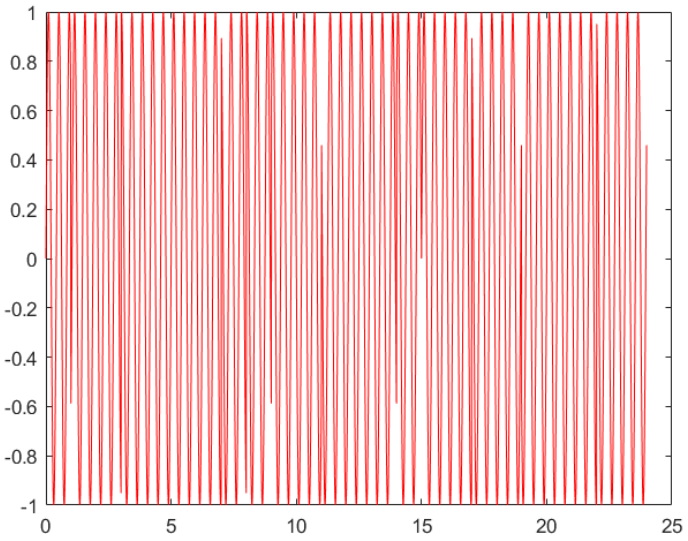
Wykres 2
x(t)



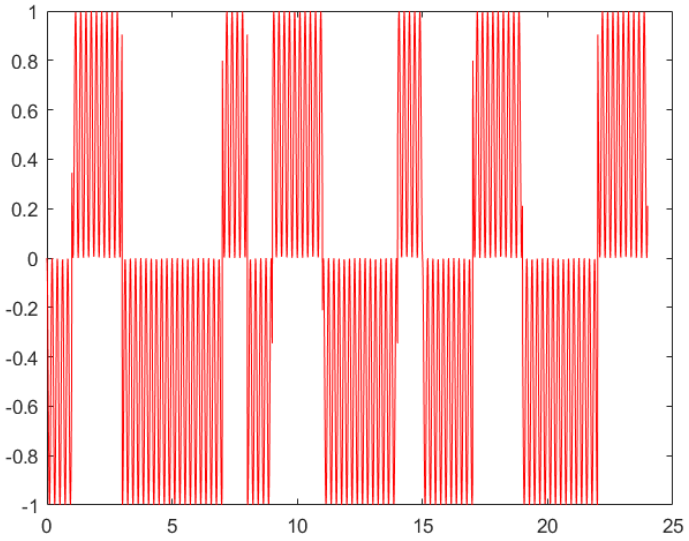
Wykres 3
p(t)



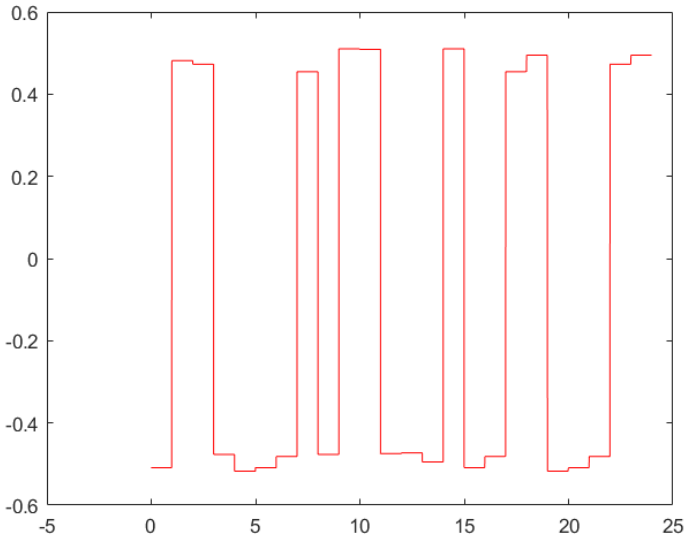
Wykres 4
m'(t)



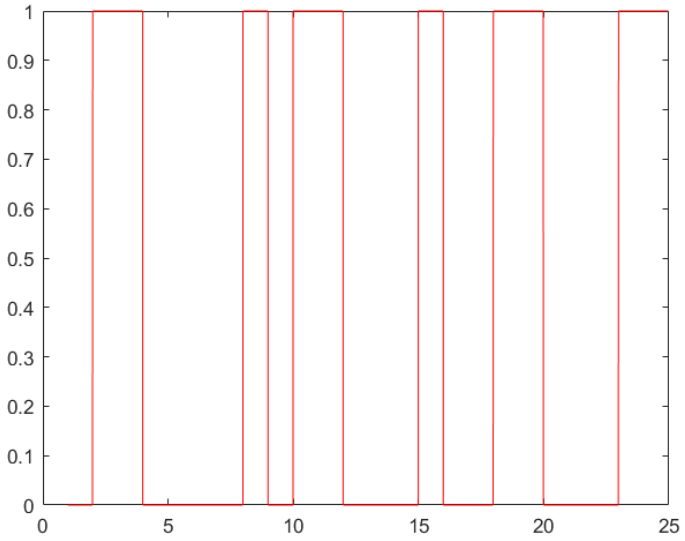
Wykres 5
zP(t)



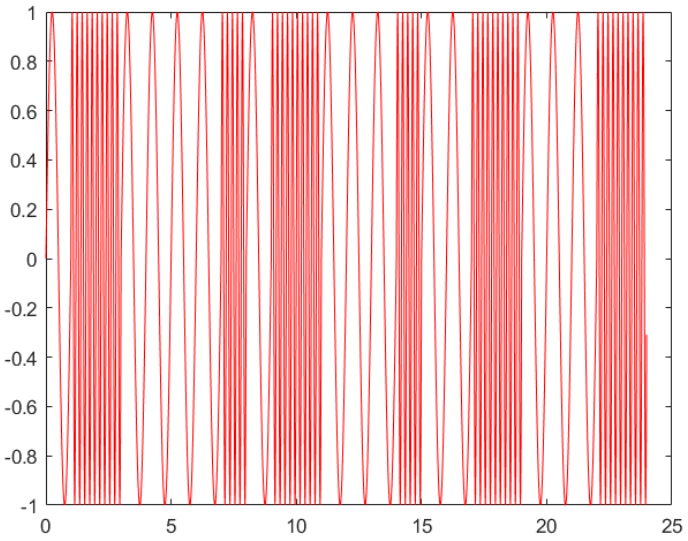
Wykres 6
x(t)



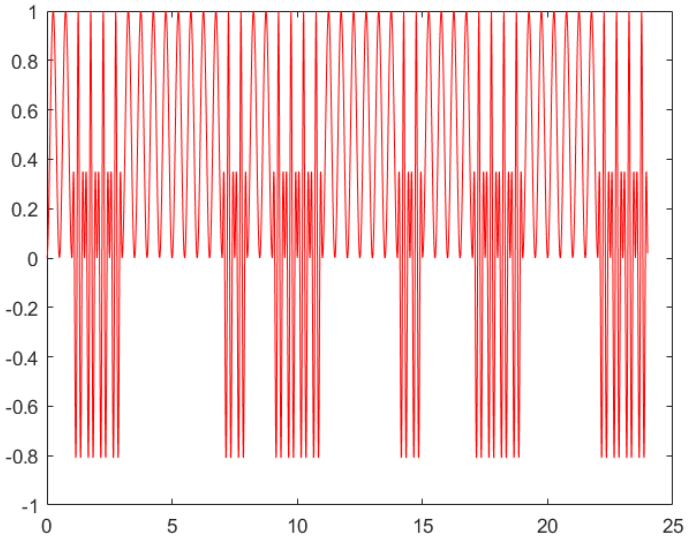
Wykres 7
p(t)



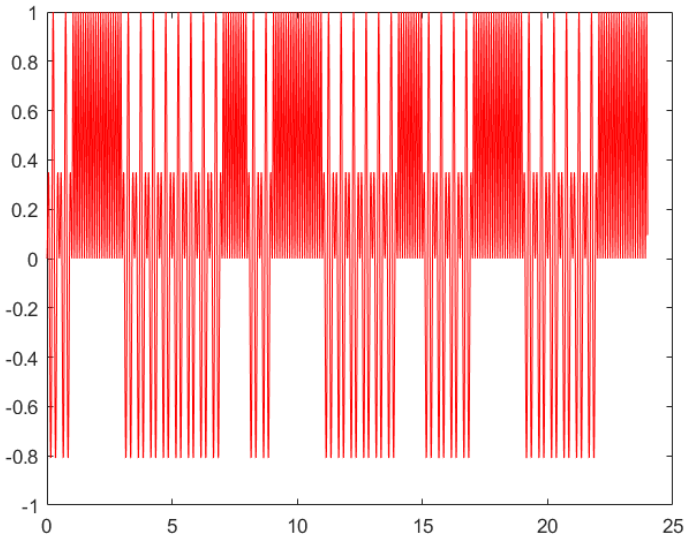
Wykres 8
m'(t)



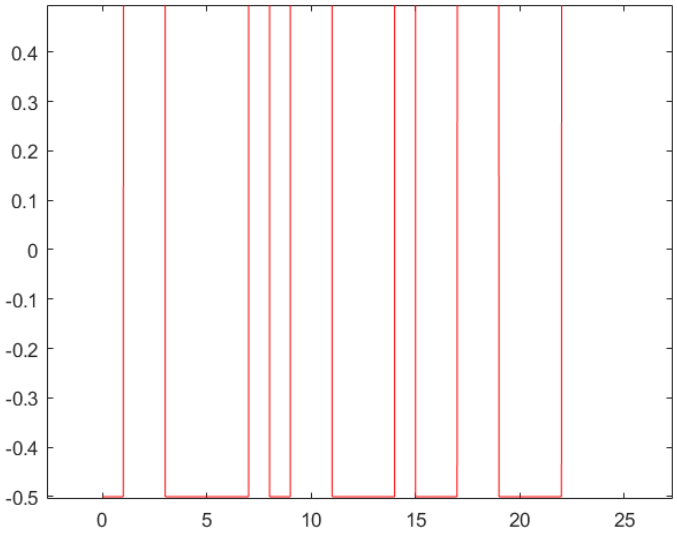
Wykres 9
zF(t)



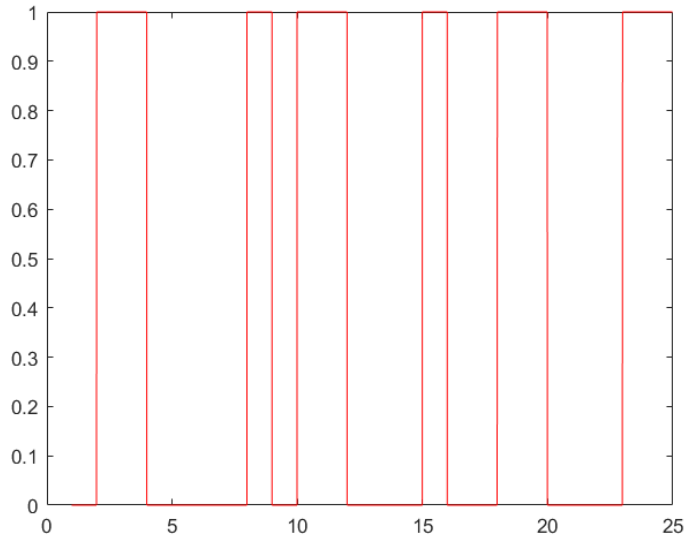
Wykres 10
x1(t)



Wykres 11
x2(t)



Wykres 12
p(t)



Wykres 13
m'(t)