Laboratorium nr 7

Wykonaj w formie programistycznej implementacji poniżej przedstawione zadania.

1) Napisz funkcję generującą sygnał zegarowy, będący sygnałem prostokątnym o zadanej częstotliwości.

```
LabSeries Lab7::genCLK(double freq, double from, double to, int steps)
{
    QVector<double> x;
    QVector<double> y;
    double range = from-to;
    if(range<0)
        range = 1;
    double step = range/steps;
    double bitStep = (1/freq)/2;
    for(int i=0; i<steps; i++)
    {
        x.append(i*step);
        if((static_cast<int>(i*step/bitStep))%2)
            y.append(0);
        else
            y.append(1);
    }
    return LabSeries(x, y, "CLK");
}
```

2) Jako generatora TTL użyj kodu generującego sygnał informacyjny $m(t)$ z tematu laboratoryjnego „5. Modulacja dyskretna". Wykorzystaj do wygenerowania sygnału $m(t)$ dwa bajty.

```
LabSeries Lab7::modTTL(LabSeries clock, QBitArray bits)
{
    QVector<double> y;
    QVector<double> x;
    bool high = true;
    for(int i=0, bit=0; i<clock.yVec.length();i++)
    {
        if(clock.yVec.at(i) == 1 && high == false)
        {
            high = true;
            bit++;
        }
        if(clock.yVec.at(i) == 0 && high == true)
        {
            high = false;
        }
        if(bit>=bits.count())
            y.append(0);
        else
            y.append(bits.at(bit));
        x.append(clock.xVec.at(i));
    }
    return LabSeries(x, y, "TTL");
}
```

Zrobiłem to na podstawie zegara tak jak inne funkcję.

3) Zgodnie z regułami przedstawionymi w skrócie z teorii napisz funkcje/programy generujące przebiegi sygnałów TTL, BAMI, NRZI i Manchaster.

Manchester:

```
LabSeries Lab7::modManchester(LabSeries clock, QBitArray bits)
{
    QVector<double> y;
    QVector<double> x;

    bool falling = false;
    bool rising = false;
    int prevClock = round(clock.yVec.first());
    int mod = 0;
    for(int i=0, bit=0; i<clock.yVec.length();i++)
    {
        int clk = round(clock.yVec.at(i));
        if(prevClock==0 && clk == 1)
            rising = true;
        if(prevClock==1 && clk == 0)
            falling = true;
        if(prevClock==0 && clk == 1)
            bit++;
        if(bit<bits.count())
        {
            if(falling && bits.at(bit))
                mod = -1;
            if(falling && !bits.at(bit))
                mod = 1;
            if(bit>0)
                if(rising && (bits.at(bit)==bits.at(bit-1)))
                    mod = (mod == 1) ? -1 : 1;
        }
        else
            mod=0;

        prevClock = clk;
        rising = false;
        falling = false;
        y.append(mod);
        x.append(clock.xVec.at(i));
    }
    return LabSeries(x, y, "Manchester");
}
```

NRZI:

```
LabSeries Lab7::modNRZI(LabSeries clock, QBitArray bits)
{
    QVector<double> y;
    QVector<double> x;

    int prevClock = round(clock.yVec.first());
    int mod = 0;
    for(int i=0, bit=0; i<clock.yVec.length();i++)
    {
        int clk = round(clock.yVec.at(i));
        if(prevClock==1 && clk == 0 && bit<bits.count())
        {
            if(bits.at(bit))
                mod = (mod >= 0) ? -1 : 1;
            bit++;
        }
        prevClock = clk;
        y.append(mod);
        x.append(clock.xVec.at(i));
    }
    return LabSeries(x, y, "NRZI");
}
```

BAMI:

```
LabSeries Lab7::modBAMI(LabSeries clock, QBitArray bits)
{
    QVector<double> y;
    QVector<double> x;

    int prevClock = round(clock.yVec.first());
    int mod = 0;
    bool counter = 0;
    for(int i=0, bit=0; i<clock.yVec.length() && bit<bits.count()+1;i++)
    {
        int clk = round(clock.yVec.at(i));
        if(prevClock==1 && clk == 0 && bit<bits.count())
        {
            if(bits.at(bit))
            {
                if(counter)
                {
                    mod = -1;
                    counter = false;
                }
                else
                {
                    mod = 1;
                    counter = true;
                }
            }
            else
                mod = 0;
            bit++;
        }
        prevClock = clk;
        y.append(mod);
        x.append(clock.xVec.at(i));
    }
    return LabSeries(x, y, "BAMI");
}
```

4) Napisz dekodery dla kodów TTL, BAMI, NRZI i Manchaster.
Przetestuj poprawność ich działania.

Dekoder TTL:

```
QBitArray Lab7::decTTL(int clockFreq, LabSeries mod)
{
    QBitArray bits;
    double step = (1/static_cast<double>(clockFreq));
    double x = step/2;
    bits.fill(false, static_cast<int>((mod.xVec.last()-mod.xVec.first())/x));
    int bit = 0;
    for(int i=0; x<mod.xVec.last(); i++)
    {
        if(mod.xVec.at(i)>x)
        {
            x+=step;
            bits.setBit(bit, mod.yVec.at(i)==1);
            bit++;
        }
    }
    bits.resize(bit);
    return bits;
}
```

Dekoder Manchester:

```
QBitArray Lab7::decManchester(int clockFreq, LabSeries mod)
{
    QBitArray bits;
    double step = (1/static_cast<double>(clockFreq))/2;
    double x = step/2;
    bits.fill(false, static_cast<int>((mod.xVec.last()-mod.xVec.first())/x));
    int bit = 0;
    bool cycle = false;
    int first = 0;
    int second = 0;
    for(int i=0; x<mod.xVec.last(); i++)
    {
        if(mod.xVec.at(i)>x)
        {
            if(cycle==false)
            {
                first = round(mod.yVec.at(i));
                cycle = true;
                x+=step;
            }
            else
            {
                second = round(mod.yVec.at(i));
                bool newBit = first >= 0 && second <= 0;
                bits.setBit(bit, newBit);
                bit++;
                cycle = false;
                x+=step;
            }
        }
    }
    bits.resize(bit);
    return bits;
}
```

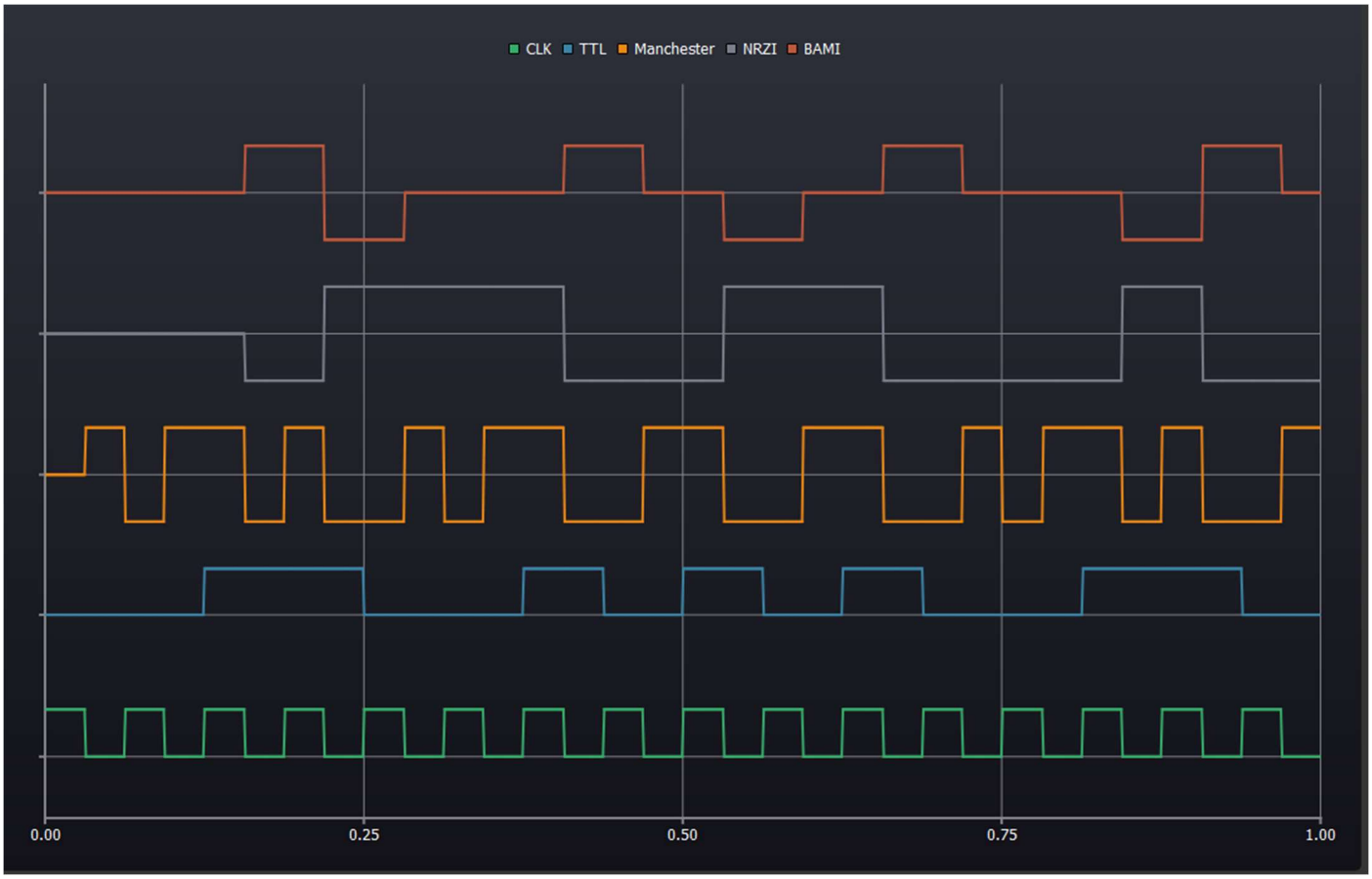
Dekoder NRZI:

```
QBitArray Lab7::decNRZI(int clockFreq, LabSeries mod)
{
    QBitArray bits;
    double step = (1/static_cast<double>(clockFreq));
    double x = step*0.75;
    bits.fill(false, static_cast<int>((mod.xVec.last()-mod.xVec.first())/x));
    int prevBit = 0;
    int bit = 0;
    for(int i=0; x<mod.xVec.last(); i++)
    {
        if(mod.xVec.at(i)>x)
        {
            x+=step;
            bool newBit = mod.yVec.at(i) != prevBit;
            prevBit = mod.yVec.at(i);
            bits.setBit(bit, newBit);
            bit++;
        }
    }
    bits.resize(bit);
    return bits;
}
```

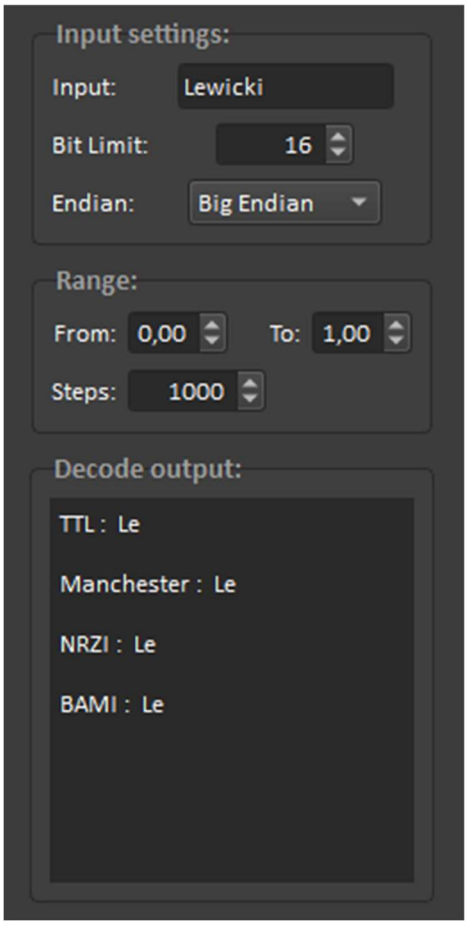
Dekoder BAMI:

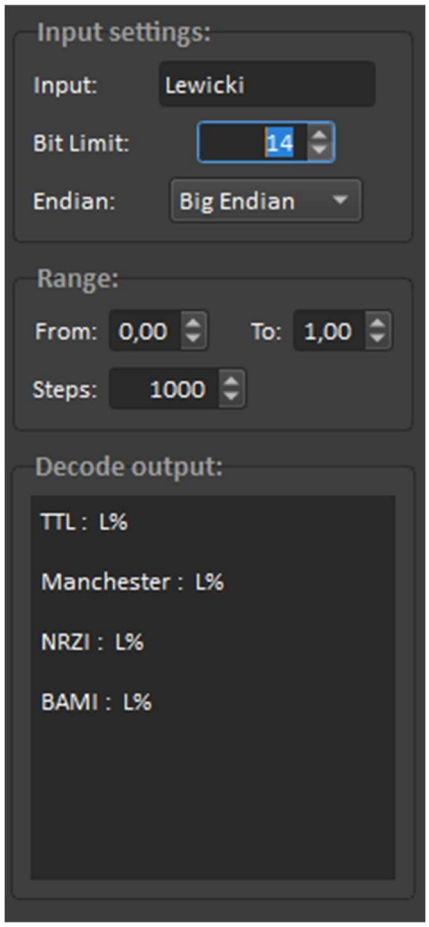
```
QBitArray Lab7::decBAMI(int clockFreq, LabSeries mod)
{
    QBitArray bits;
    double step = (1/static_cast<double>(clockFreq));
    double x = step*0.75;
    bits.fill(false, static_cast<int>((mod.xVec.last()-mod.xVec.first())/x));
    int bit = 0;
    for(int i=0; x<mod.xVec.last(); i++)
    {
        if(mod.xVec.at(i)>x)
        {
            x+=step;
            bool newBit = mod.yVec.at(i) != 0;
            bits.setBit(bit, newBit);
            bit++;
        }
    }
    bits.resize(bit);
    return bits;
}
```

Wykresy sygnałów:



Input kodera i output dekodera:





Przy niepełnych bajtach resultat dekodowania zawiera błędne znaki. Zgodnie z założeniem.