# Laboratorium nr 8

1) Zaimplementuj funkcję kodującą kodem Hamming (7,4) zadany strumień binarny. Do generowania strumienia binarnego użyj funkcji **S2BS** napisanej na laboratoriach „5. Modulacja dyskretna”.

2) Napisz funkcję negującą wskazany numer bitu w strumieniu binarnym z zadania pierwszego.

W funkcji zawarta jest od razu funkcjonalność pozwalająca na zanegowanie do 7 losowych bitów w każdym zakodowanym bajcie.

```
QBitArray Lab7_8_9::encodeHamming_4bit(QBitArray bits, int error)
{
    //if the array of bits is not divisible by 4 we ignore the remainder
    //as it does not form proper byte anyway
    QBitArray encodedBits;
    encodedBits.fill(0, bits.count()*2);
    if(bits.count()<4)
        return encodedBits;
    for(int i=0; i<bits.count()-3; i=i+4)
    {
        int valD[] = {bits.at(i),
                      bits.at(i+1),
                      bits.at(i+2),
                      bits.at(i+3)
                     };
        QGenericMatrix<1, 4, int> d(valD);

        int valG[] = {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
                     };
        QGenericMatrix<4, 7, int> G(valG);
        auto h = G * d;
        auto hT = h.transposed()%2;
        int p4 = 0;
        for(int j=0; j < 7; j++)
        {
            if(hT(0,j))
                encodedBits.setBit(j+i*2);
            p4+=hT(0,j);
        }
        encodedBits.setBit(7+i*2, p4 % 2);

        QVector<int> erroredBits;
        erroredBits <<0<<1<<2<<3<<4<<5<<6<<7;
        std::random_shuffle(erroredBits.begin(), erroredBits.end());
        for(int j=0; j<error; j++)
        {
            encodedBits.toggleBit(erroredBits.at(j)+i*2);
        }
    }
    qDebug()<<"Encoded Data with"<<error<<"error(s) per byte: "<<encodedBits;
    return encodedBits;
}
```

3) Zaimplementuj funkcję dekodującą kod Hamminga (7,4), sprawdź poprawność działania.

4) Zaimplementuj rozwinięcie (koder i dekoder) kodu Hamming (7,4) jako kod SECDED, posiadający dodatkowy bit parzystości. Zweryfikuj działanie poprzez zanegowanie dwóch bitów w strumieniu binarnym.

```
QBitArray Lab7_8_9::decHamming_4bit(QBitArray bits)
{
    //if we introduce more than 2 errors this algorithm will not work

    QBitArray decodedBits;
    decodedBits.fill(0, bits.count()/2-1);
    if(bits.count()<8)
        return decodedBits;
    for(int i=0; i<bits.count()-7; i=i+8)
    {
        int valD[] = {bits.at(i),
                      bits.at(i+1),
                      bits.at(i+2),
                      bits.at(i+3),
                      bits.at(i+4),
                      bits.at(i+5),
                      bits.at(i+6),
                     };
        QGenericMatrix<1, 7, int> d(valD);

        int valH[] = {1, 0, 1, 0, 1, 0, 1,
                      0, 1, 1, 0, 0, 1, 1,
                      0, 0, 0, 1, 1, 1, 1
                     };

        QGenericMatrix<7, 3, int> H(valH);
        auto p = (H * d) % 2;
        bool discard = false;
        int n = p(0,0)*1 + p(1,0)*2 + p(2,0)*4;
        if(n)
        {
            valD[n-1] = valD[n-1]==0 ? 1 : 0;
            int p4 = (valD[0]+valD[1]+valD[2]+valD[3]+valD[4]
                      +valD[5]+valD[6])%2;
            if(static_cast<bool>(p4) != bits.at(i+7))
                discard = true;
        }
        //if there are more bad bits we discard the result and return 0000
        if(!discard)
        {
            int res[4] = {valD[2], valD[4], valD[5], valD[6]};
            for(int j=0; j<4; j++)
                if(res[j])
                    decodedBits.setBit(j+i/2);
        }
    }
    qDebug()<<"Decoded Data: "<<decodedBits<<"\n";
    return decodedBits;
}
```

**Weryfikacja działania:**

INPUT: „TRANSMISJA DANYCH”

Wersja bez błędnych bitów:

```
INPUT:  QBitArray(0010 1010 0100 1010 1000 0010 0111 0010 1100 1010 1011 0010 1001 0010 1100 1010 0101 0010 1000 0010 0000 0100 0010
0010 1000 0010 0111 0010 1001 1010 1100 0010 0001 0010)

Encoded Data with 0 error(s) per byte:  QBitArray(0101 0101 1011 0100 1001 1001 1011 0100 1110 0001 0101 0101 0001 1110 0101 0101
0111 1000 1011 0100 0110 0110 0101 0101 0011 0011 0101 0101 0111 1000 1011 0100 0100 1011 0101 0101 1110 0001 0101 0101 0000 0000
1001 1001 0101 0101 0101 0101 1110 0001 0101 0101 0001 1110 0101 0101 0011 0011 1011 0100 0111 1000 0101 0101 1101 0010 0101 0101)

Decoded Data:  QBitArray(0010 1010 0100 1010 1000 0010 0111 0010 1100 1010 1011 0010 1001 0010 1100 1010 0101 0010 1000 0010 0000
0100 0010 0010 1000 0010 0111 0010 1001 1010 1100 0010 0001 0010 0000 0000 0000 00)
```

(dodatkowe zera na końcu są spowodowane tym, że sygnał przechodzący przez tor transmisyjny jest dłuższy niż ilość bitów)

Wersja z 1 błędnym bitem w każdym zakodowanym bajcie:

```
INPUT:  QBitArray(0010 1010 0100 1010 1000 0010 0111 0010 1100 1010 1011 0010 1001 0010 1100 1010 0101 0010 1000 0010 0000 0100 0010
0010 1000 0010 0111 0010 1001 1010 1100 0010 0001 0010)

Encoded Data with 1 error(s) per byte:  QBitArray(0001 0101 1010 0100 1001 1011 1011 0000 1110 0011 0101 1101 0011 1110 1101 0101
0111 1001 1011 1100 0110 1110 0100 0101 0011 1011 0101 0001 0011 1000 1010 0100 0100 1001 0101 1101 1111 0001 0001 0101 0000 0100
1001 1101 0101 1101 0101 0111 1010 0001 0100 0101 0011 1110 0101 0100 0011 1011 1011 0110 0111 0000 0001 0101 1111 0010 0101 0111)

Decoded Data:  QBitArray(0010 1010 0100 1010 1000 0010 0111 0010 1100 1010 1011 0010 1001 0010 1100 1010 0101 0010 1000 0010 0000
0100 0010 0010 1000 0010 0111 0010 1001 1010 1100 0010 0001 0010 0000 0000 0000 00)
```

Wersja z 2 błędnymi bitami w każdym zakodowanym bajcie:

```
INPUT:  QBitArray(0010 1010 0100 1010 1000 0010 0111 0010 1100 1010 1011 0010 1001 0010 1100 1010 0101 0010 1000 0010 0000 0100 0010
0010 1000 0010 0111 0010 1001 1010 1100 0010 0001 0010)

Encoded Data with 2 error(s) per byte:  QBitArray(0110 0101 1011 1110 1000 1101 1001 0101 0110 0000 1101 1101 0011 1010 0101 1100
1111 1010 1011 1110 0111 1110 0100 0111 0001 0111 0001 0100 0111 1110 0011 1100 0000 1111 0100 0100 1100 0011 1101 0100 1000 0001
0000 1001 0001 0100 0111 0001 1010 1001 0101 0000 0011 1111 0100 0111 1011 0010 1010 0101 0110 1001 0111 0111 0100 0010 1101 1101)

Decoded Data:  QBitArray(0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 00)
```