Laboratorium nr 6

1) Wykorzystaj sygnały $z_A(t)$, $z_P(t)$ i $z_F(t)$ z poprzednich zajęć laboratoryjnych.
2) Zaimplementuj powyżej przedstawione demodulatory.
3) Na podstawie obserwacji sygnału $p(t)$ dobierz eksperymentalnie wartość progu h.

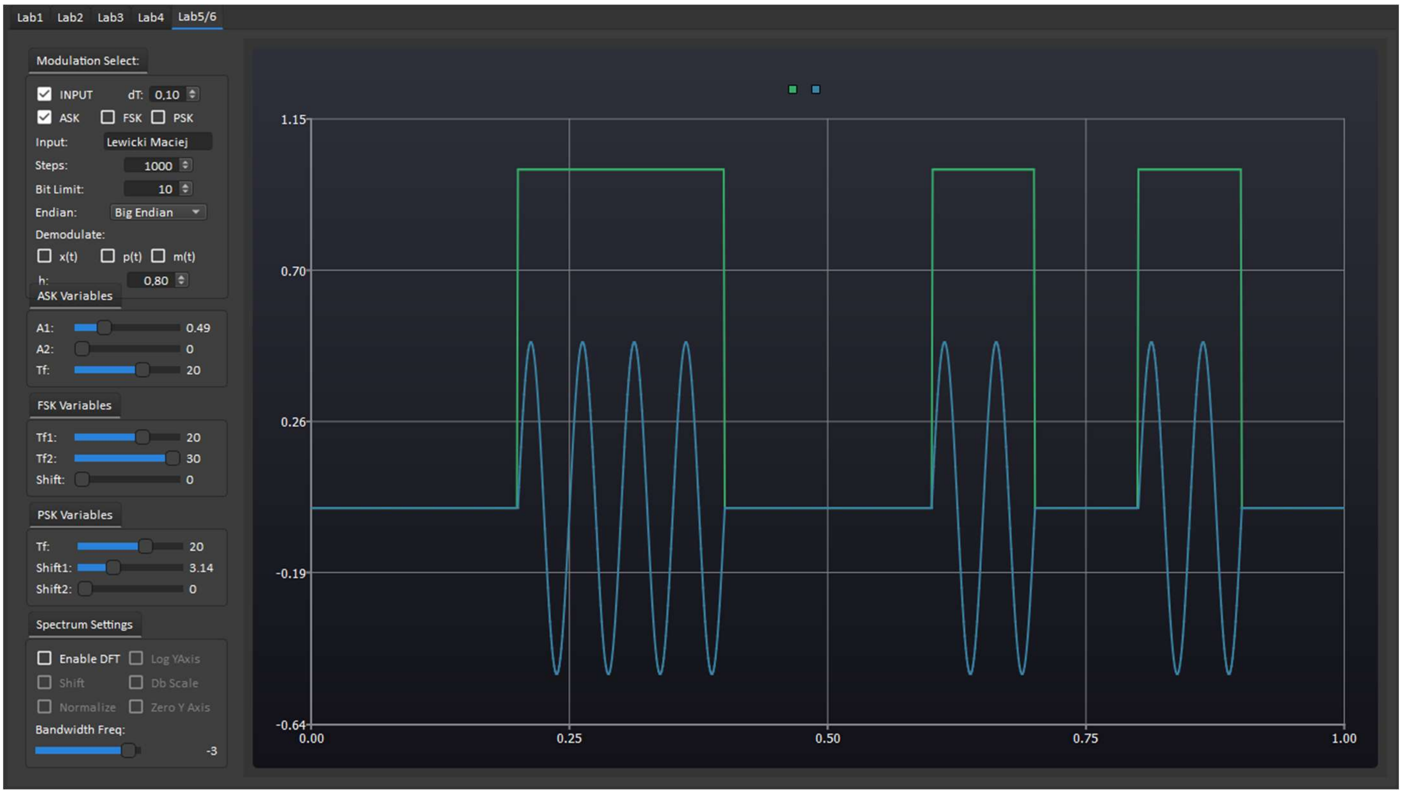
W wyniku porównania z wartością progową na wyjściu komparatora wygenerować odpowiedź postaci:

$$\overline{m}'(t) = \begin{cases} 0 & p(t) < h \\ 1 & p(t) \geq h \end{cases}$$

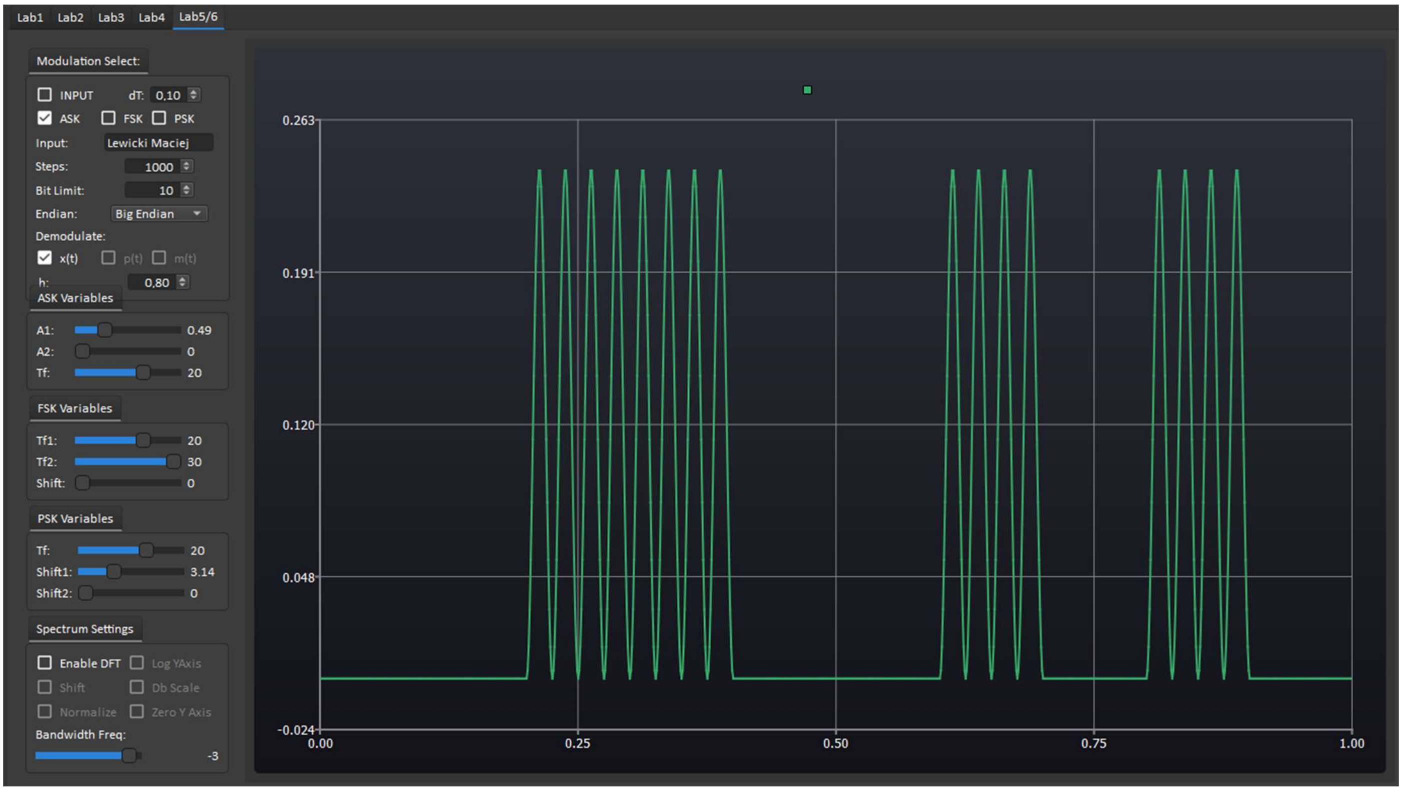
Modulacja i demodulacja ASK:

```
if(ask->checkState()==2)
{
    //during demodulation increasing A2 requires also increasing h
    QVector<double> tempY;
    QLineSeries* series = new QLineSeries(this);
    for(int i=0; i<stepsVal; i++)
    {
        //modulating binary signal
        if(vecY.at(i))
            tempY.append((askAmp1_l)*(sin(2*M_PI*askTargetFreq_l*vecX.at(i)+askShift_l)));
        else
            tempY.append((askAmp2_l)*(sin(2*M_PI*askTargetFreq_l*vecX.at(i)+askShift_l)));
    }
    if(dX->checkState()==2 || dP->checkState()==2 || dM->checkState()==2)
    {
        //multiplication of modulated signal and carrier signal
        for(int i=0; i<tempY.length(); i++)
            tempY[i] *= (askAmp1_l)*(sin(2*M_PI*askTargetFreq_l*vecX.at(i)+askShift_l));
    }
    if(dP->checkState()==2 || dM->checkState()==2)
    {
        for(int b=0; b<bitLim; b++)
        {
            int bitLength = stepsVal/bitLim;
            int bitStartStep = b*bitLength;
            int bitEndStep = (b+1)*bitLength;
            double integral = 0;

            for(int i=bitStartStep; i<bitEndStep-1; i++)
            {
                //trapezoid integral with step based on the same delta as series
                //                 (a      +      b)        /2*            h
                integral += (tempY.at(i)+tempY.at(i+1))/2*(vecX.at(i+1)-vecX.at(i));


                //values included in example charts had this integral multiplied by 1000 so im doing the same
                //this was probably done in order to avoid setting h to tiny numbers eg. h = 0.000001
                tempY[i] = integral*1000;
            }
        }
    }
    if(dM->checkState()==2)
    {
        //setting demodulated signal values based on h threshold
        for(int i=0; i<tempY.length(); i++)
            if(tempY.at(i)>dh->value())
                tempY[i] = 1;
            else
                tempY[i] = 0;
    }
    for(int i=0; i<vecX.length() && i<tempY.length(); i++)
        series->append(vecX.at(i), tempY.at(i));
    chartView->chart()->addSeries(series);
}
```

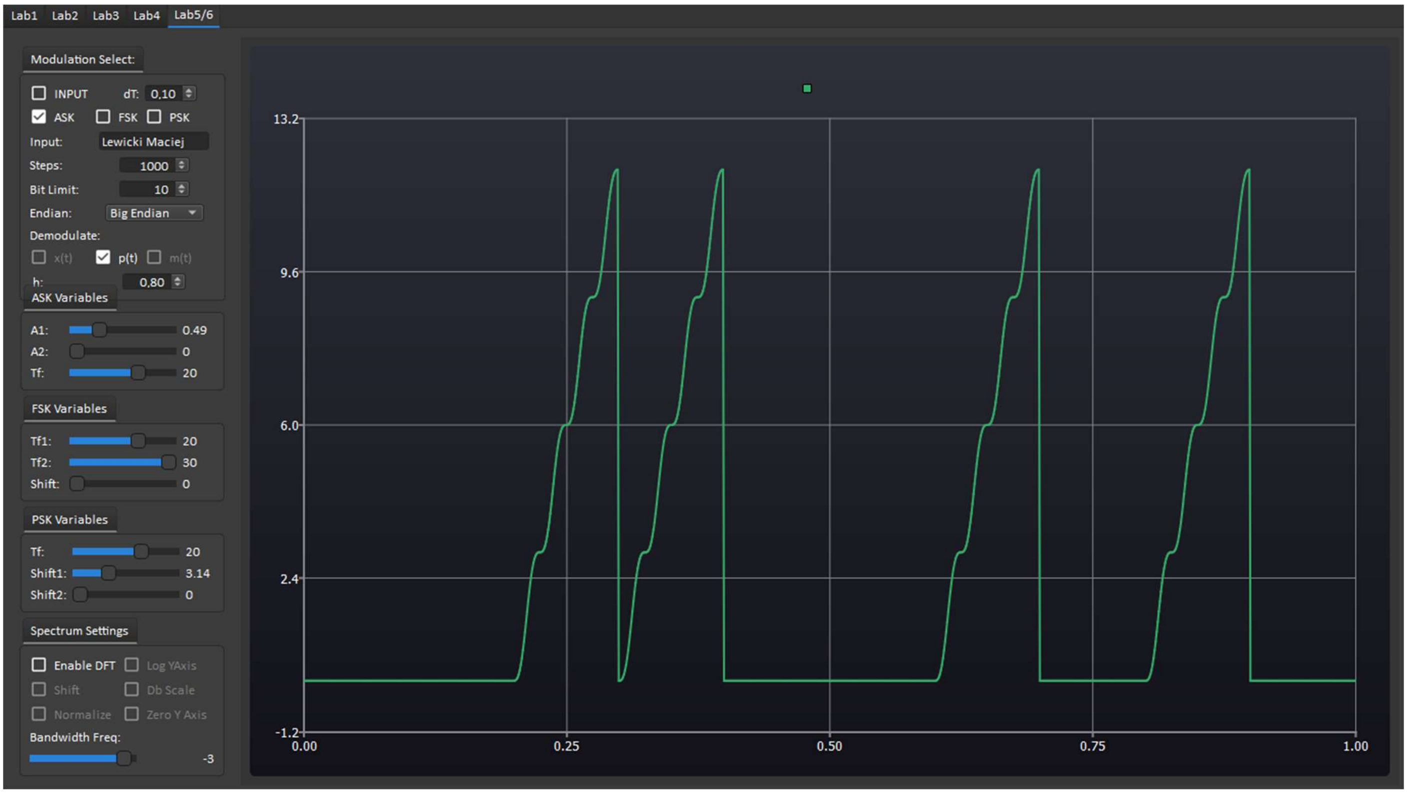
Sygnał wejściowy (10 bitów) + sygnał zmodulowany ASK:



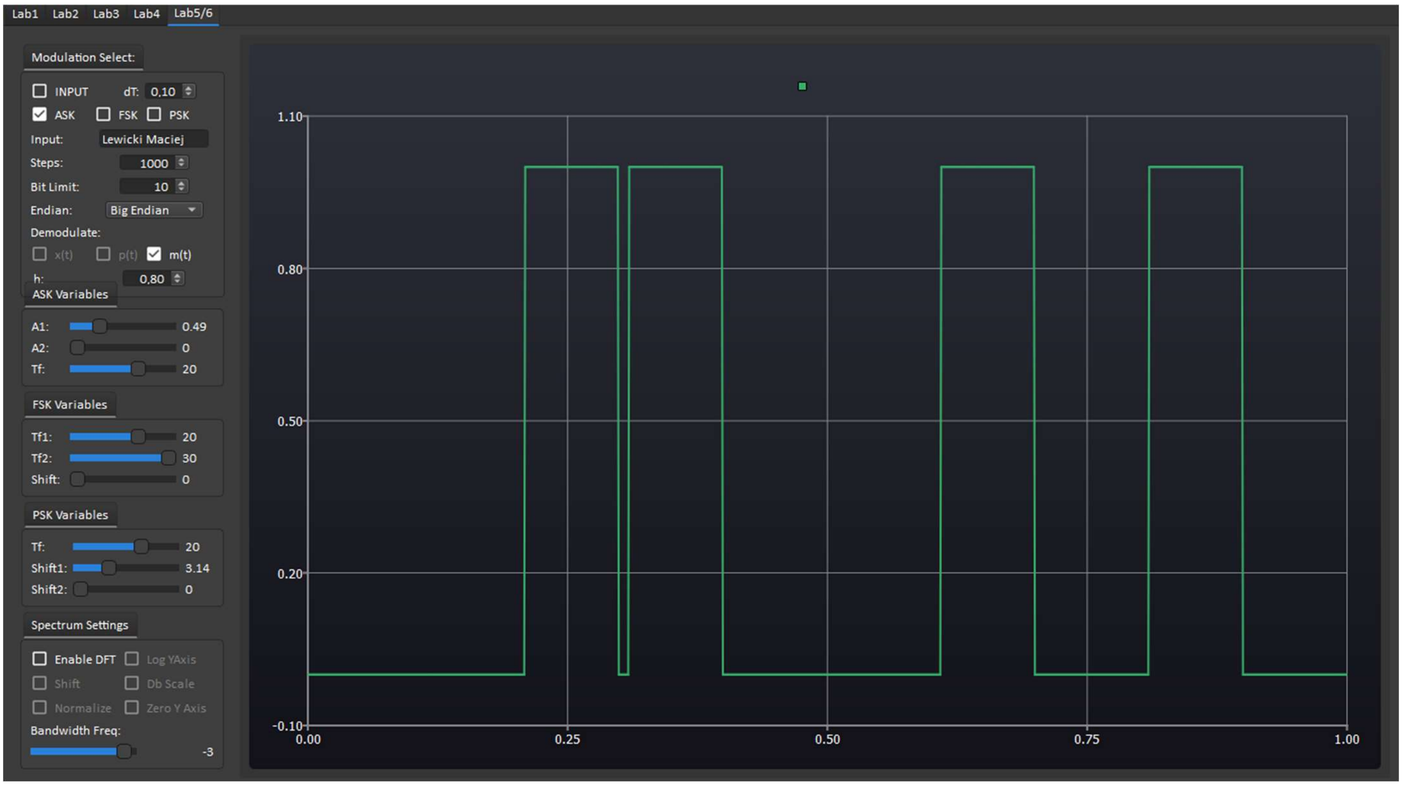
Demodulacja ASK x(t):



Demodulacja ASK p(t):



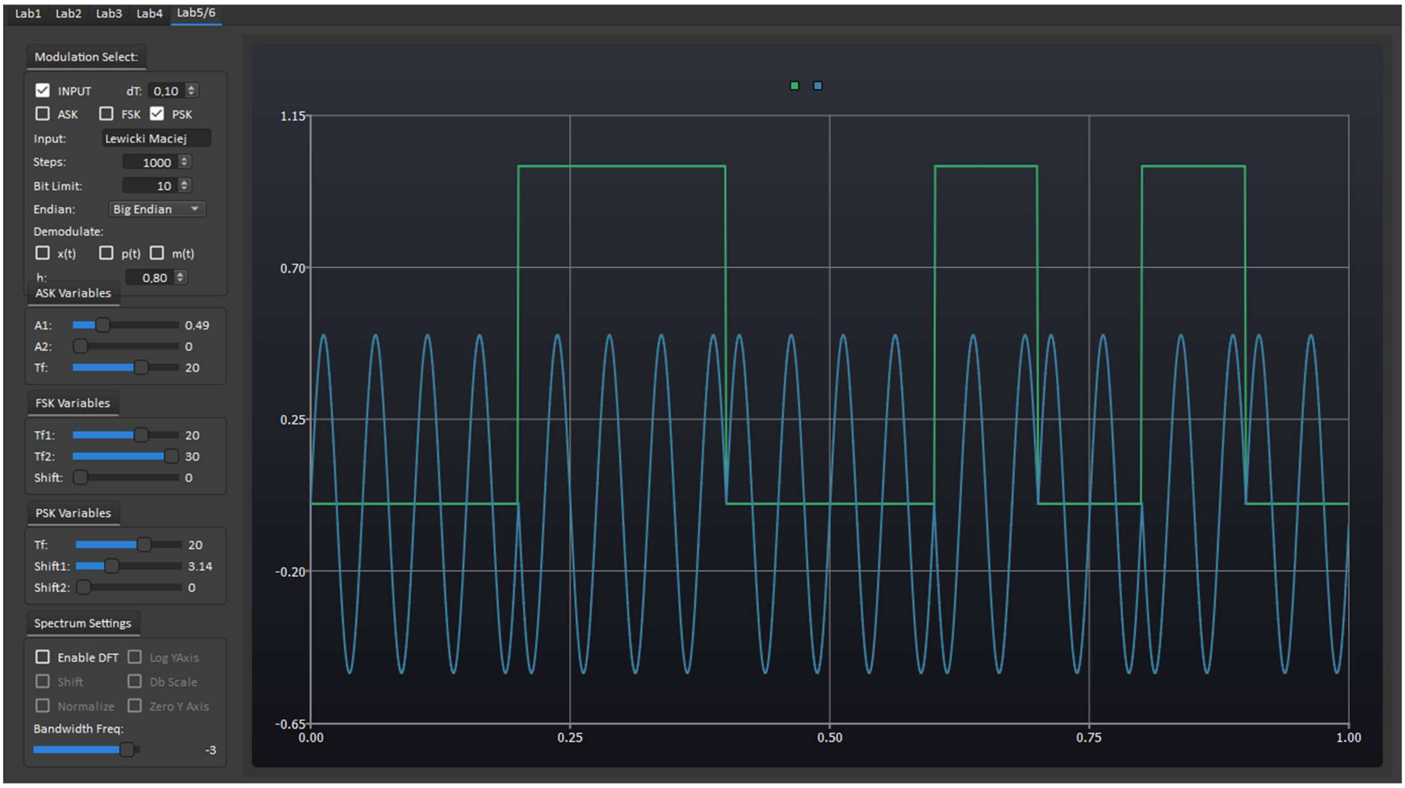
Demodulacja ASK m(t):



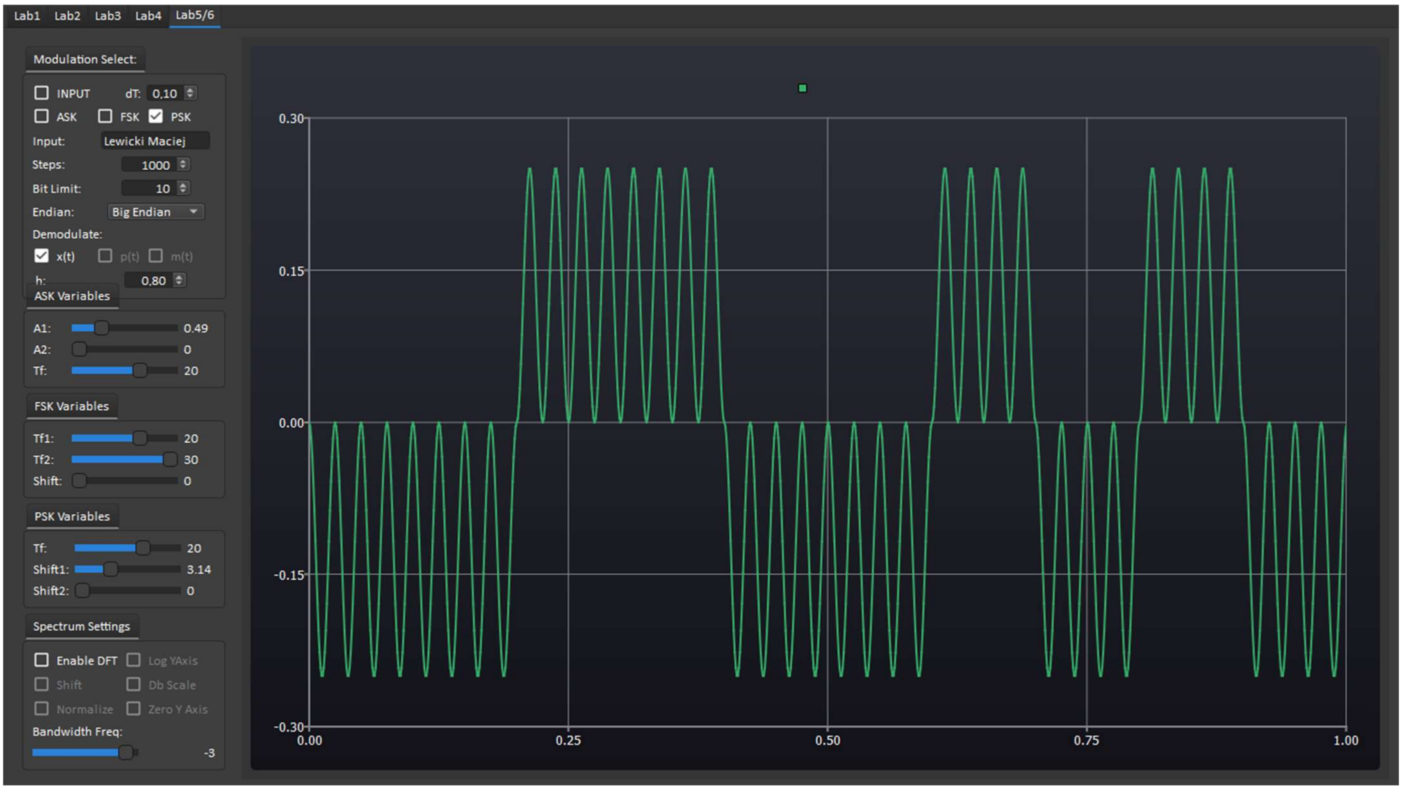
Modulacja i demodulacja PSK:

```
if(psk->checkState()==2)
{
    QVector<double> tempY;
    QLineSeries* series = new QLineSeries(this);
    for(int i=0; i<stepsVal; i++)
    {
        if(vecY.at(i))
            tempY.append((pskAmp_l)*(sin(2*M_PI*pskTargetFreq_l*vecX.at(i)+pskShift1_l)));
        else
            tempY.append((pskAmp_l)*(sin(2*M_PI*pskTargetFreq_l*vecX.at(i)+pskShift2_l)));
    }
    if(dX->checkState()==2 || dP->checkState()==2 || dM->checkState()==2)
    {
        for(int i=0; i<tempY.length(); i++)
            tempY[i] *= (pskAmp_l)*(sin(2*M_PI*pskTargetFreq_l*vecX.at(i)+pskShift1_l));
    }
    if(dP->checkState()==2 || dM->checkState()==2)
    {
        for(int b=0; b<bitLim; b++)
        {
            int bitLength = stepsVal/bitLim;
            int bitStartStep = b*bitLength;
            int bitEndStep = (b+1)*bitLength;
            double integral = 0;
            for(int i=bitStartStep; i<bitEndStep-1; i++)
            {
                //                (a      +      b)        /2*             h
                integral += (tempY.at(i)+tempY.at(i+1))/2*(vecX.at(i+1)-vecX.at(i));
                tempY[i] = integral*1000;
            }
        }
    }
    if(dM->checkState()==2)
    {
        for(int i=0; i<tempY.length(); i++)
            if(tempY.at(i)>dh->value())
                tempY[i] = 1;
            else
                tempY[i] = 0;
    }
    for(int i=0; i<vecX.length() && i<tempY.length(); i++)
        series->append(vecX.at(i), tempY.at(i));
    chartView->chart()->addSeries(series);
}
```

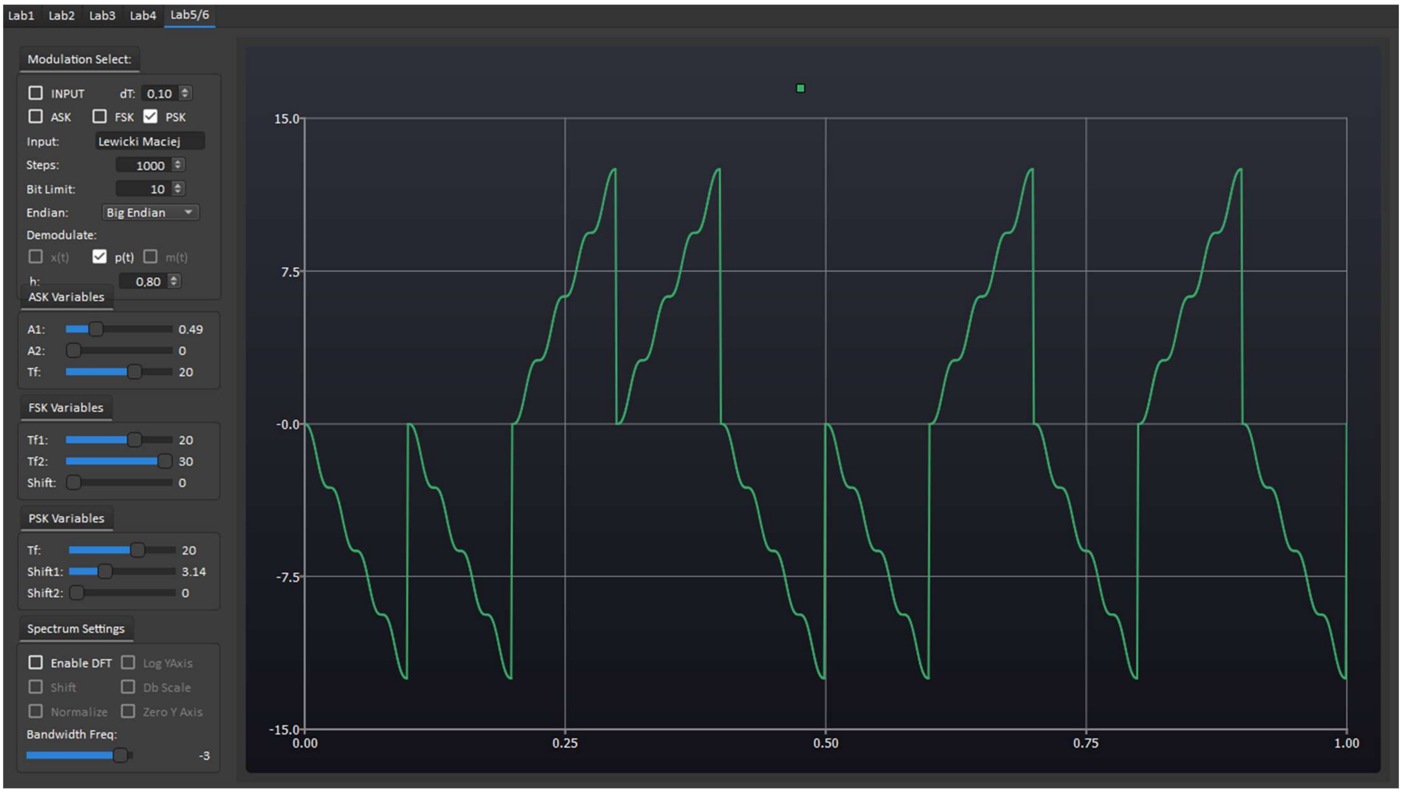
Sygnał wejściowy (10 bitów) + sygnał zmodulowany PSK:



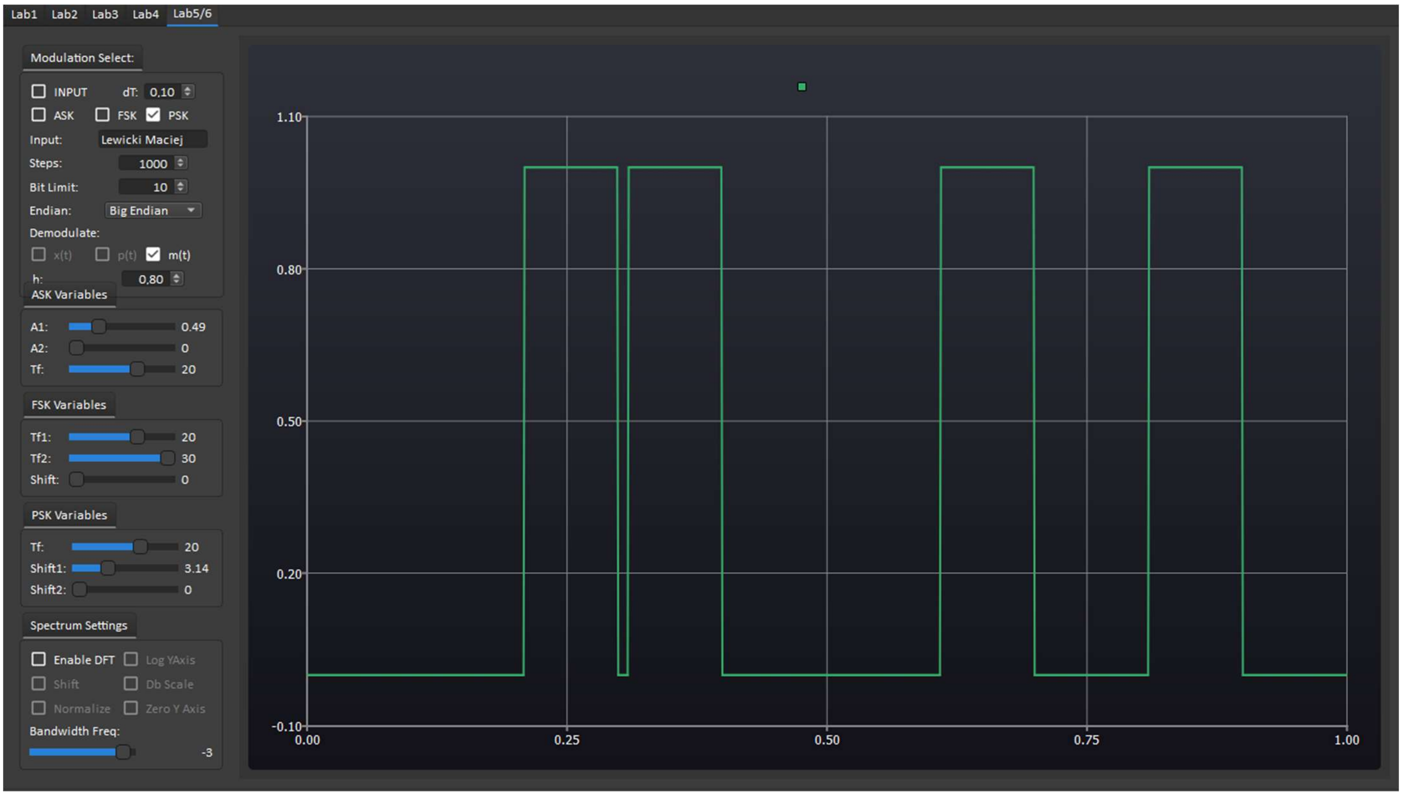
Demodulacja PSK x(t):



Demodulacja PSK p(t):



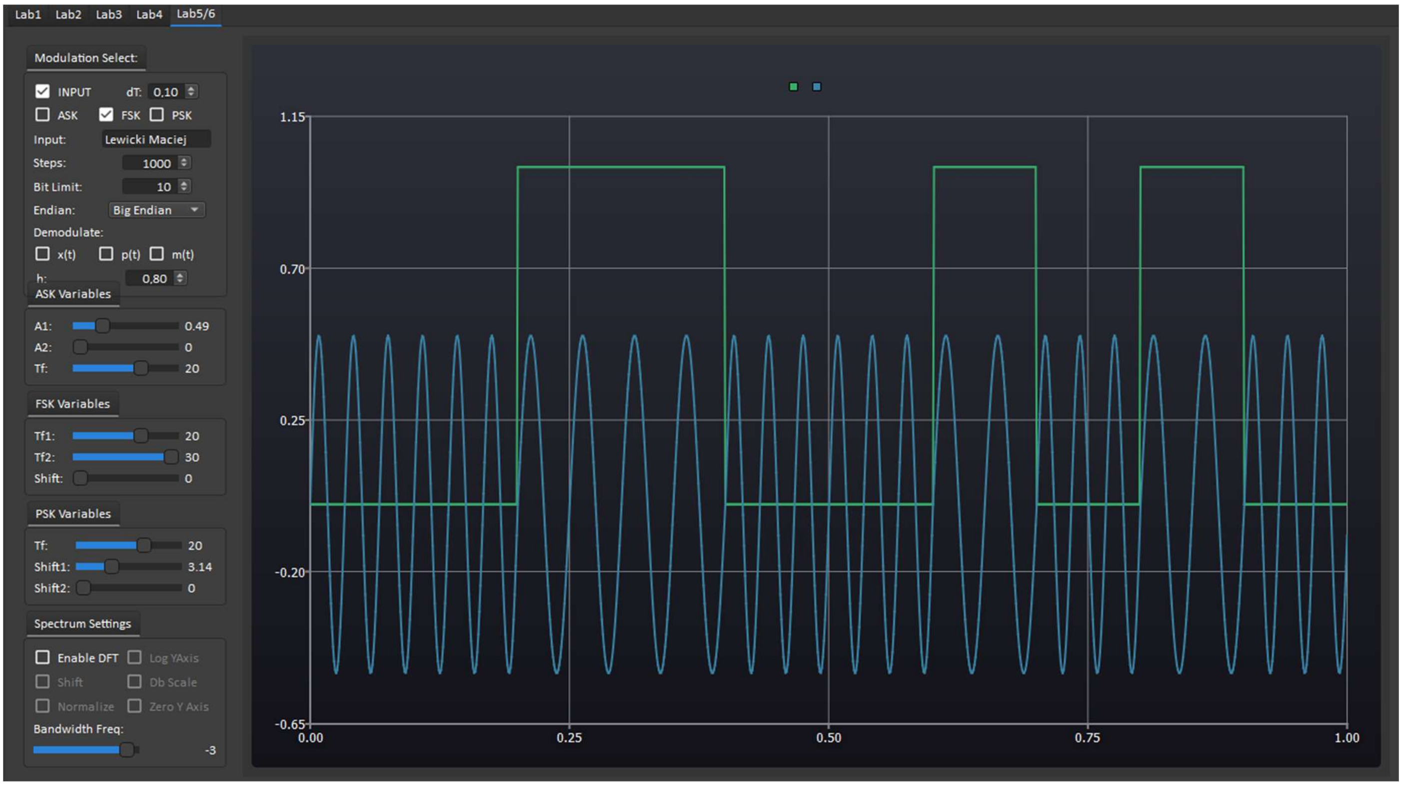
Demodulacja PSK m(t):



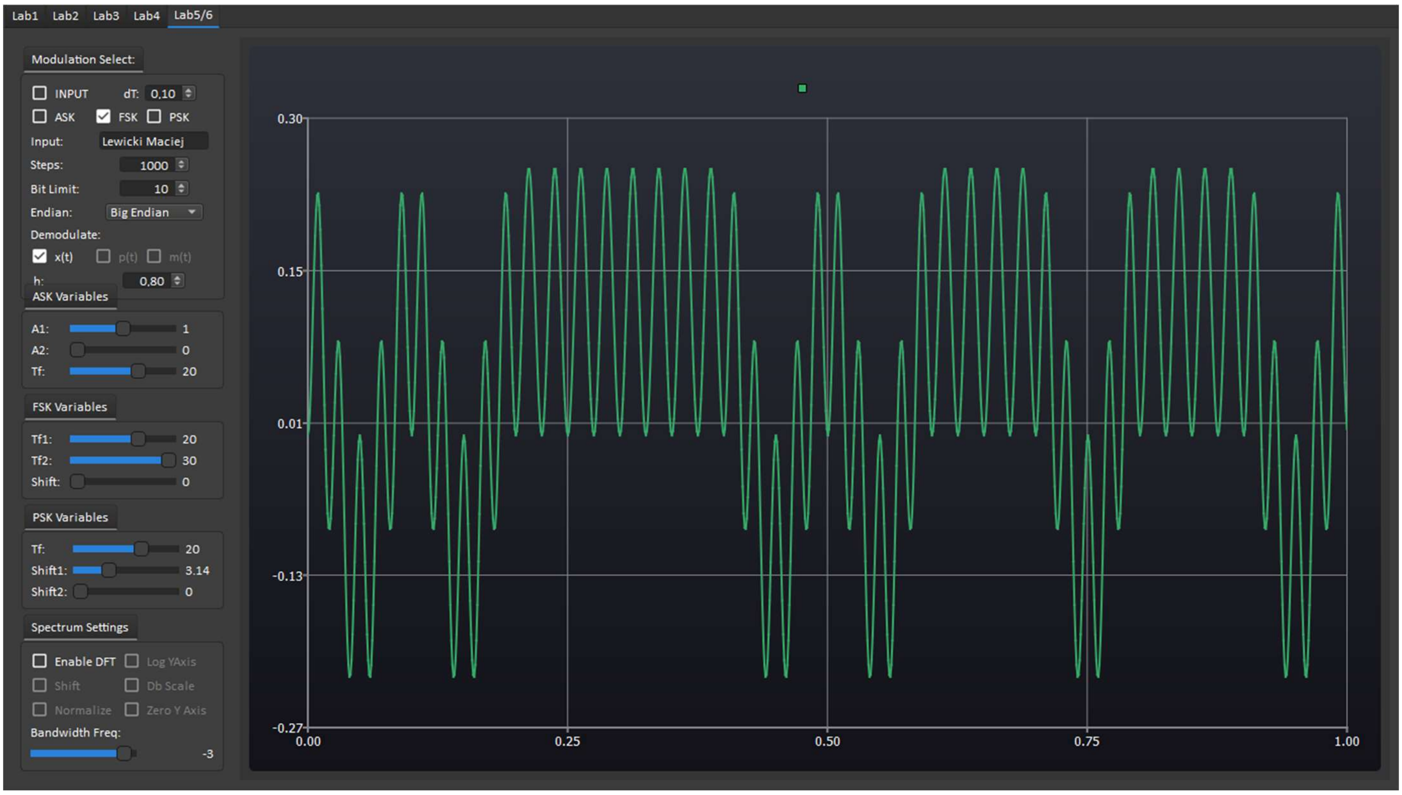
Modulacja i demodulacja FSK:

```
if(fsk->checkState()==2)
{
    QVector<double> tempY;
    QVector<double> tempY2;
    QVector<double> tempY3;
    QLineSeries* series = new QLineSeries(this);
    for(int i=0; i<stepsVal; i++)
    {
        if(vecY.at(i))
            tempY.append((fskAmp_l)*(sin(2*M_PI*fskTargetFreq1_l*vecX.at(i)+fskShift_l)));
        else
            tempY.append((fskAmp_l)*(sin(2*M_PI*fskTargetFreq2_l*vecX.at(i)+fskShift_l)));
    }
    if(dX->checkState()==2 || dP->checkState()==2 || dM->checkState()==2)
    {
        //multiplication of modulated signal and two carrier signals into separate arrays
        for(int i=0; i<tempY.length(); i++)
            tempY2.append(tempY.at(i)*(fskAmp_l)*(sin(2*M_PI*fskTargetFreq2_l*vecX.at(i)+fskShift_l)));
        for(int i=0; i<tempY.length(); i++)
            tempY[i] *= (fskAmp_l)*(sin(2*M_PI*fskTargetFreq1_l*vecX.at(i)+fskShift_l));
    }
    if(dP->checkState()==2 || dM->checkState()==2)
    {
        for(int b=0; b<bitLim; b++)
        {
            int bitLength = stepsVal/bitLim;
            int bitStartStep = b*bitLength;
            int bitEndStep = (b+1)*bitLength;
            double integral = 0;
            double integral2 = 0;
            for(int i=bitStartStep; i<bitEndStep-1; i++)
            {
                //trapezoid integral with step based on the same delta as series
                //                (a      +      b)        /2*            h
                integral += (tempY.at(i)+tempY.at(i+1))/2*(vecX.at(i+1)-vecX.at(i));
                tempY[i] = integral*1000;

                //this is integral of the second multiplied signal
                integral2 += (tempY2.at(i)+tempY2.at(i+1))/2*(vecX.at(i+1)-vecX.at(i));
                tempY2[i] = integral2*1000;

                //final integral
                tempY3.append(integral*1000-integral2*1000);
            }
        }
    }
    if(dM->checkState()==2)
    {
        //setting demodulated signal values based on h threshold
        for(int i=0; i<tempY3.length() && i<tempY.length(); i++)
            if(tempY3.at(i)>dh->value())
                tempY[i] = 1;
            else
                tempY[i] = 0;
    }
```

Sygnał wejściowy (10 bitów) + sygnał zmodulowany FSK:

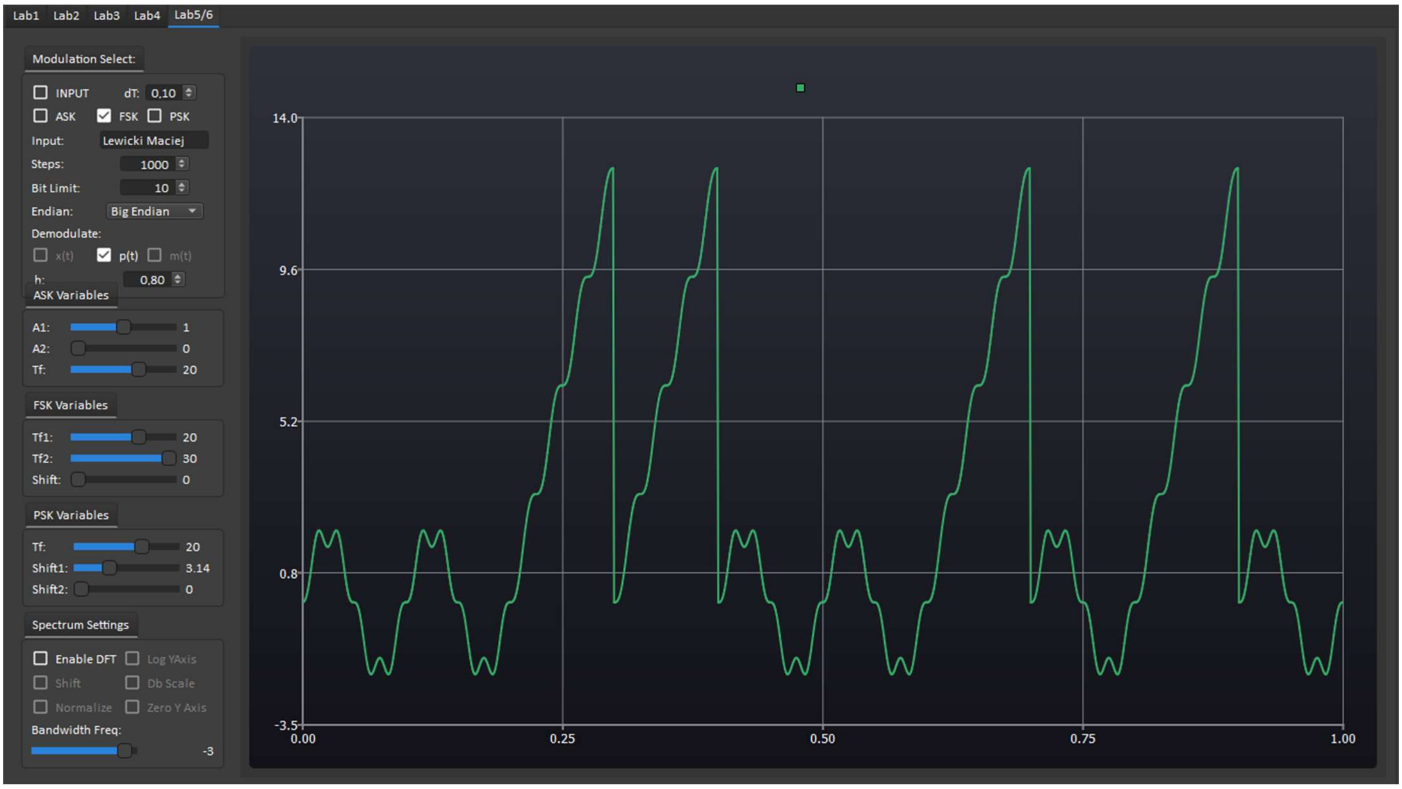


Demodulacja FSK x1(t):

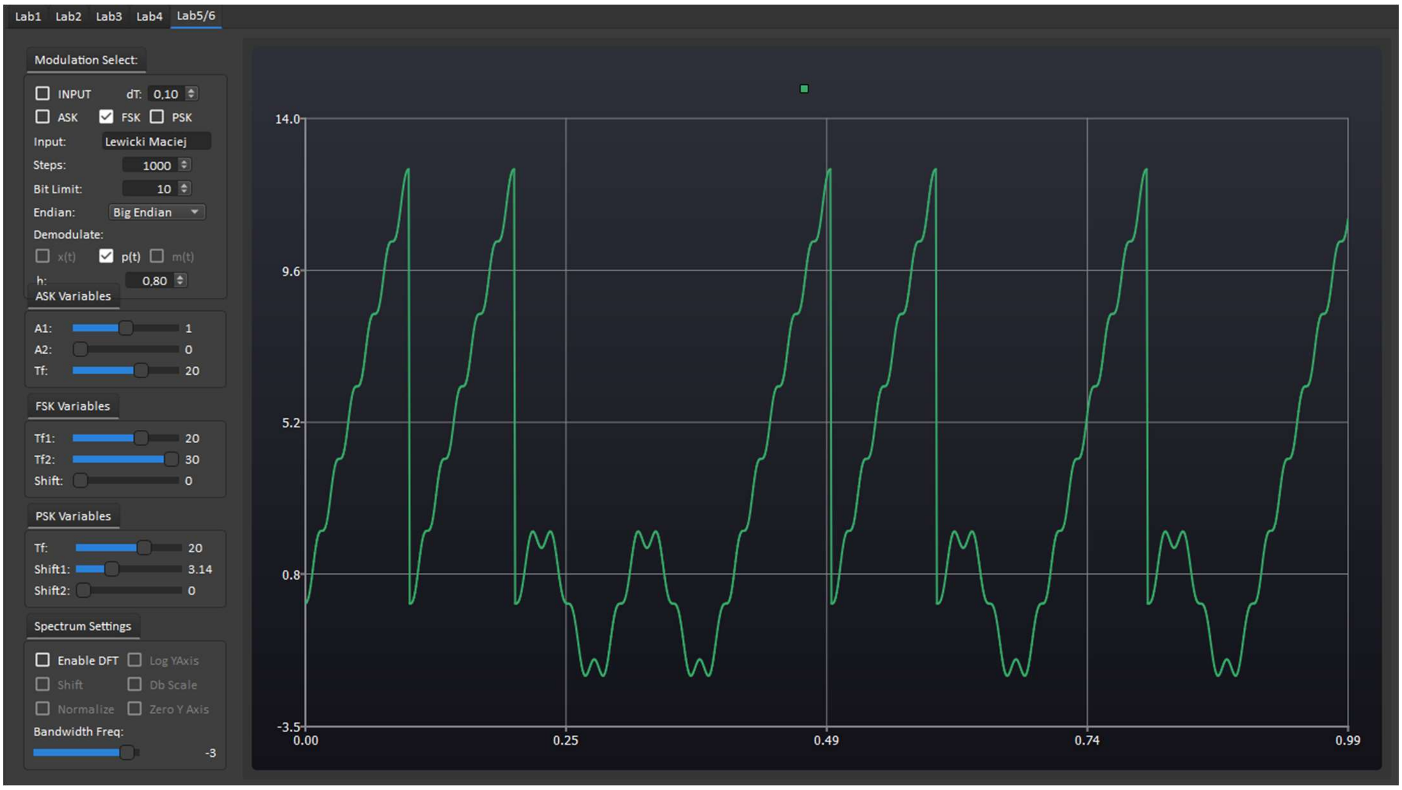


Demodulacja FSK x2(t):

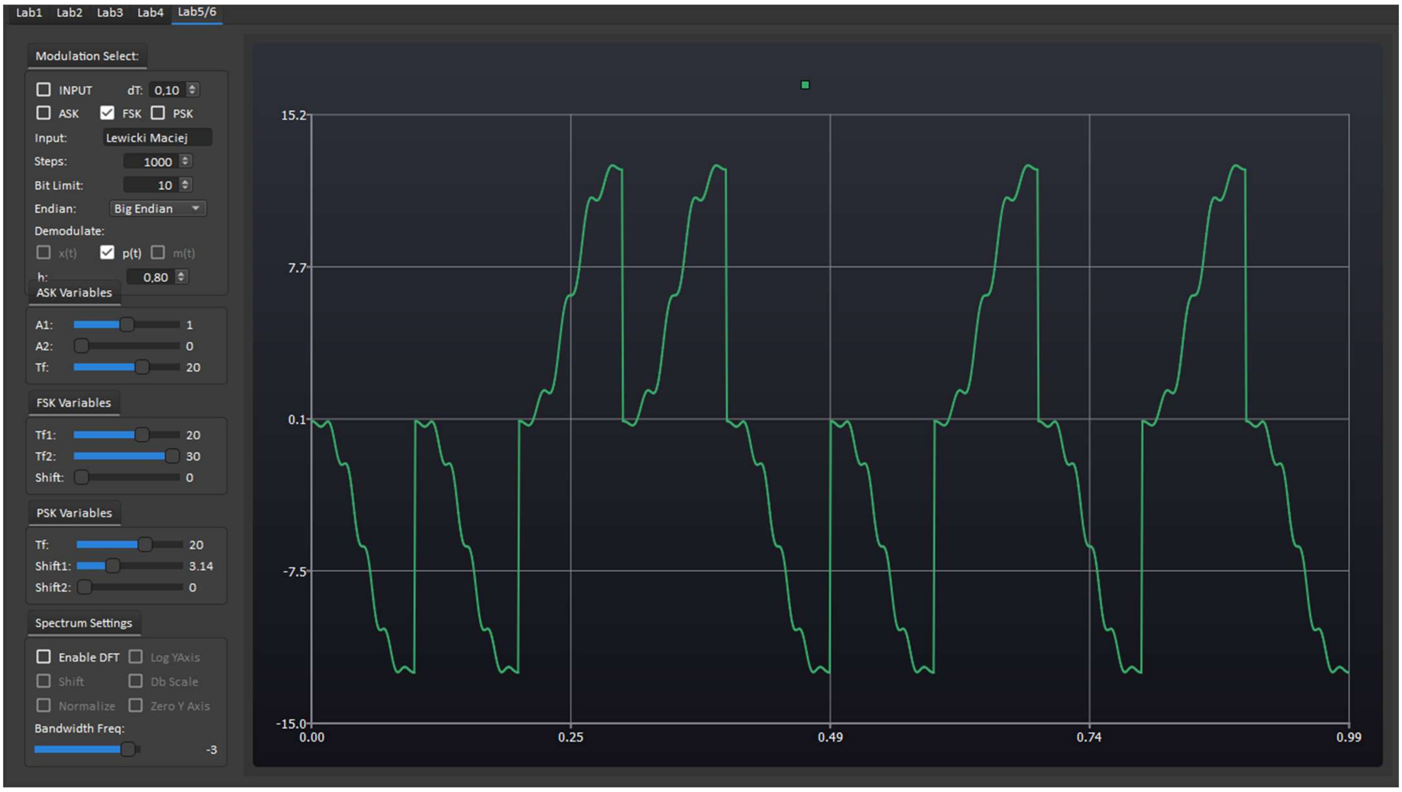
Demodulacja FSK p1(t):



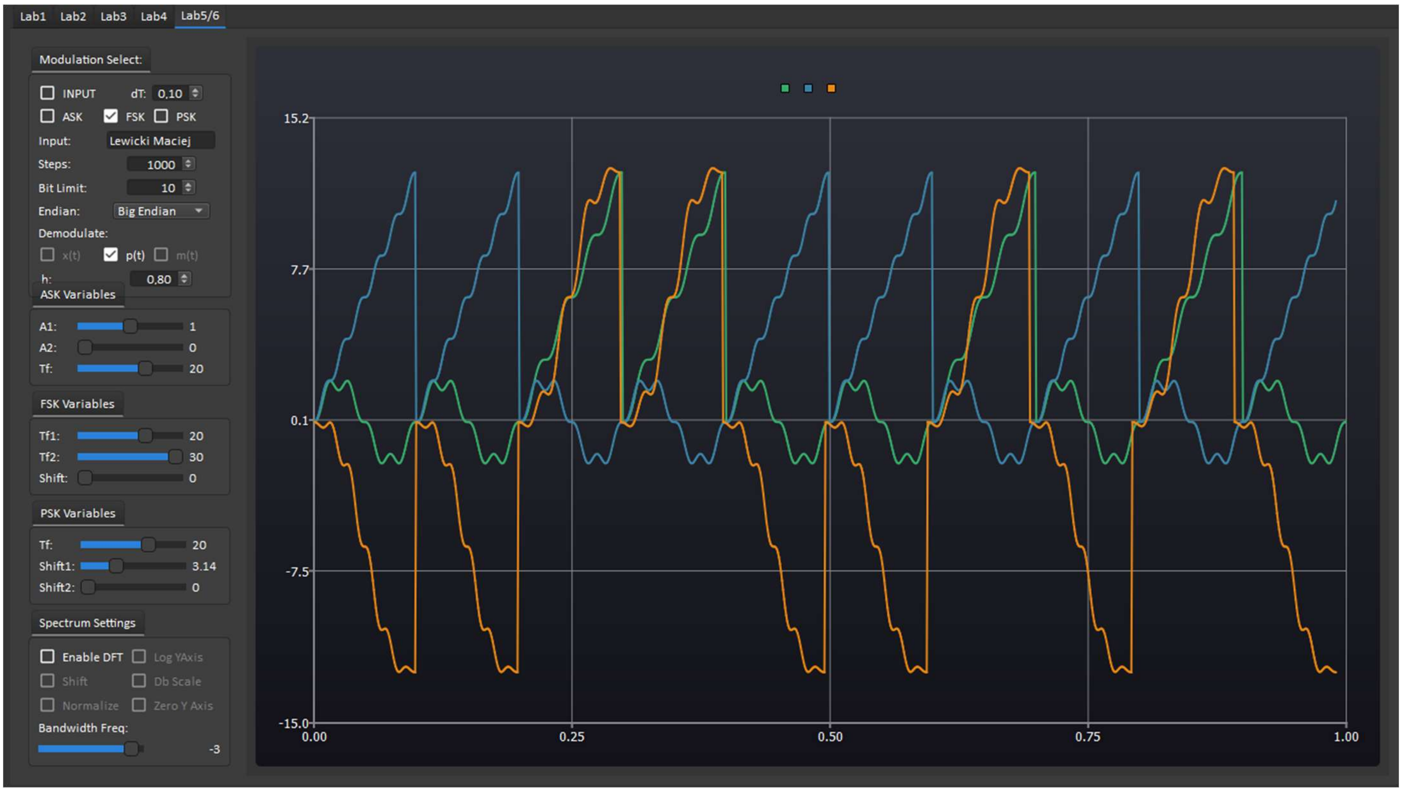
Demodulacja FSK p2(t):



Demodulacja FSK p3(t) wynikowy:



Demodulacja FSK p1(t) + p2(t) + p3(t):



Demodulacja FSK m(t):