Sprawozdanie Lab07

Autor: Kamil Szóstak

```
import math
import numpy as np
import matplotlib.pyplot as plt
import array
import math
import numpy as np
import matplotlib.pyplot as plt
import array
from Utilities import tile, linspace



def CLK(samplesPerBit, clockCount):
    halfSamples = int(samplesPerBit / 2)

    clockSamples = samplesPerBit * clockCount * [None]
    for i in range(clockCount * 2):
        clockSamples[i * halfSamples:(i + 1) * halfSamples] = tile((i % 2) == 0, halfSamples)
    return clockSamples

def TTL():
    ycords=[] ; xcords=[] ; x=0
    for i in range (len(mt)):
        xtemp=x
        x=x+0.1
        x=round(x,1)
        if(mt[i]==0):
            for j in np.linspace(1/10,2/10):
                ycords.append(0)
```

```
        else:
            for j in np.linspace(1/10,2/10):
                ycords.append(1)
    xcords=np.linspace(0,len(mt)/10,len(ycords))
    return xcords,ycords


def Manchester(clk, ttl):
    M_C = [0]
    currentValue = 0
    #wzgórze malejące
    for i in range(len(clk) - 1):
        if (clk[i] == 1 and clk[i + 1] == 0):
            if (ttl[i] == 0):
                currentValue = 1
            else:
                currentValue = -1
        #wzgórze narastające
        elif (clk[i] == 0 and clk[i + 1] == 1):
            if (ttl[i] == ttl[i + 1]):
                currentValue *= -1
        M_C.append(currentValue)
    return M_C


def DekoderManchester(clock, manchester, samplesPerbit):
    quarterSamplesPerBit = int(samplesPerbit / 4)
    clock = tile(1, quarterSamplesPerBit) + clock
    clock = clock[:int(len(clock) - quarterSamplesPerBit)]


    bits = []
    for i in range(len(clock) - 1):
        #wzgórze malejące
```

```
        if (clock[i] == 1 and clock[i + 1] == 0):
            bits.append(manchester[i])
    return bits

def NRZI():
    top = [] ; bot = [] ; final = [] ; xcords = [] ; switch=mt[1]
    for x in np.linspace(1/10,2/10):
        top.append(1)
        bot.append(0)
    for i in range (len(mt)):
        if(mt[i]==1):
            switch=1-switch
        if(switch==0):
            for x in range(len(bot)):
                final.append(bot[x])
        else:
            for x in range(len(top)):
                final.append(top[x])

    xcords=np.linspace(0,len(mt)/10,len(final))
    return xcords,final

def BAMI():
    top = [] ; bot = [] ; final = [] ; zero = [] ; switch=1
    for x in np.linspace(1/10,2/10):
        top.append(1)
        zero.append(0)
        bot.append(-1)
    for i in range (len(mt)):
        if(mt[i]==0):
            for x in range(len(zero)):
```

```
            final.append(zero[x])
        else:
            if(switch==1):
                switch=-1
                for x in range(len(bot)):
                    final.append(bot[x])
            else:
                switch=1
                for x in range(len(top)):
                    final.append(top[x])

    xcords=np.linspace(0,len(mt)/10,len(final))
    return xcords,final

def info_Signal(secondsPerBit, bits, s_p_Bit):
    time = linspace(0, secondsPerBit * len(bits), s_p_Bit * len(bits))

    s_samples = s_p_Bit * len(bits) * [None] #próbki sygnału
    for i, bit in enumerate(bits):
        s_samples[i * s_p_Bit:(i + 1) * s_p_Bit] = tile(bit, s_p_Bit)
    return time, s_samples



bity=[1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0]
clk = CLK(100, len(bity))
t, ttl = info_Signal(0.1, bity, 100)
manchesterSamples = Manchester(clk, ttl)
manchesterBits = DekoderManchester(clk, ttl, 100)
tzz, manchestersTtl = info_Signal(0.1, manchesterBits, 100)

plt.figure()
```

```
plt.subplot(411)
plt.title('CLK')
plt.plot(t, clk)
plt.subplot(412)
plt.title('TTL')
plt.plot(t, ttl)
plt.subplot(413)
plt.title('Manchester')
plt.plot(t, manchesterSamples)
plt.subplot(414)
plt.title('Manchester zdekodowany')
plt.plot(tzz, manchestersTtl)
plt.show()
```