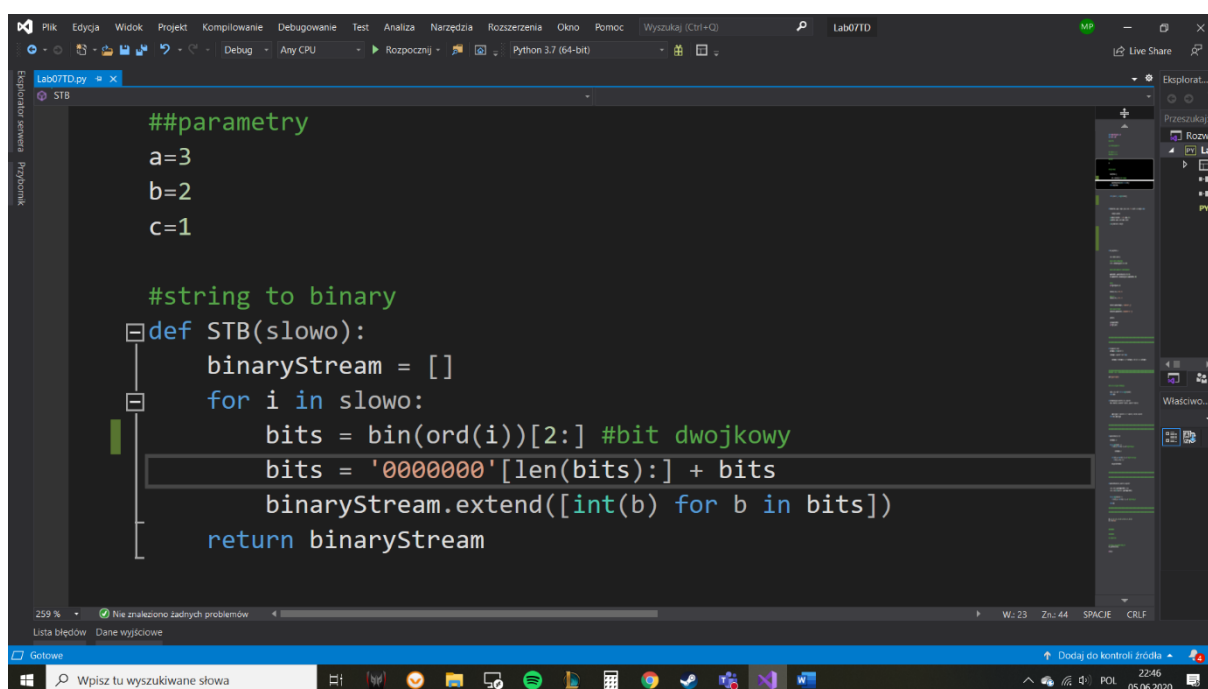


Sprawozdanie Lab07

Mateusz Proc 20 N1 nr 45123

(Do opracowania mieliśmy tylko dekodery Manchester)

1/.



The screenshot shows a Python IDE with a file named 'Lab07TD.py'. The code defines a function 'STB(slowo)' that converts a string into a binary stream. The function parameters are 'a=3', 'b=2', and 'c=1'. The function logic is as follows:

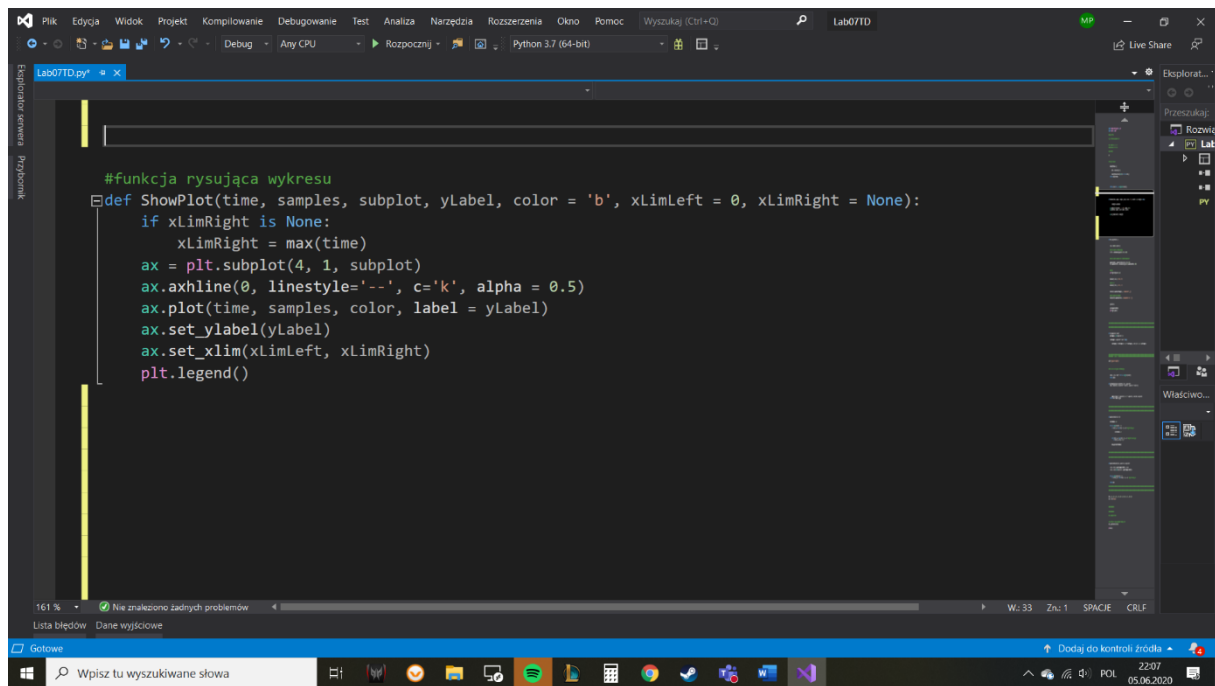
```
##parametry
a=3
b=2
c=1

#string to binary
def STB(slowo):
    binaryStream = []
    for i in slowo:
        bits = bin(ord(i))[2:] #bit dwójkowy
        bits = '000000'[len(bits):] + bits
        binaryStream.extend([int(b) for b in bits])
    return binaryStream
```

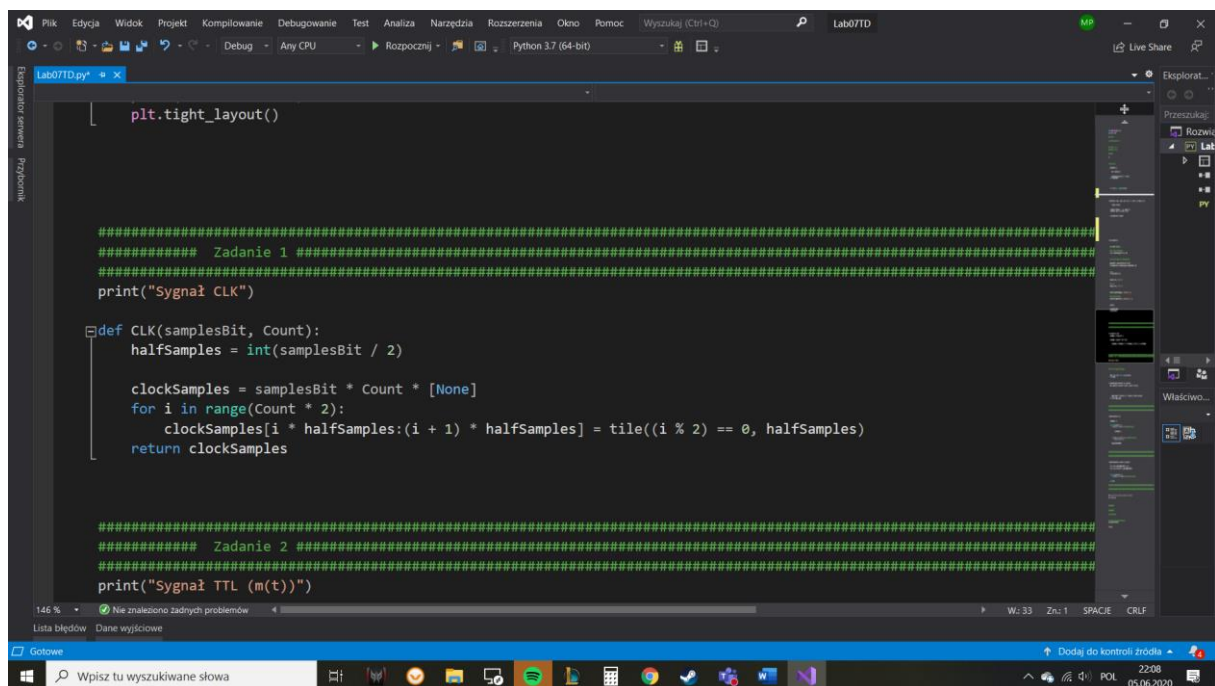
The IDE interface includes a menu bar at the top with options like 'Plik', 'Edycja', 'Widok', 'Projekt', 'Kompilowanie', 'Debugowanie', 'Test', 'Analiza', 'Narzędzia', 'Rozszerzenia', 'Okno', and 'Pomoc'. The status bar at the bottom shows '259 %', 'Nie znaleziono żadnych problemów', and 'Lista błędów Dane wyjściowe'. The taskbar at the very bottom shows various application icons and the system clock indicating '22:46' on '05.06.2020'.

Funkcja String To Binary, którą testuje później dekodery.

Z tej funkcji przekazuje dwa bajty do wygenerowania sygnału informacyjnego

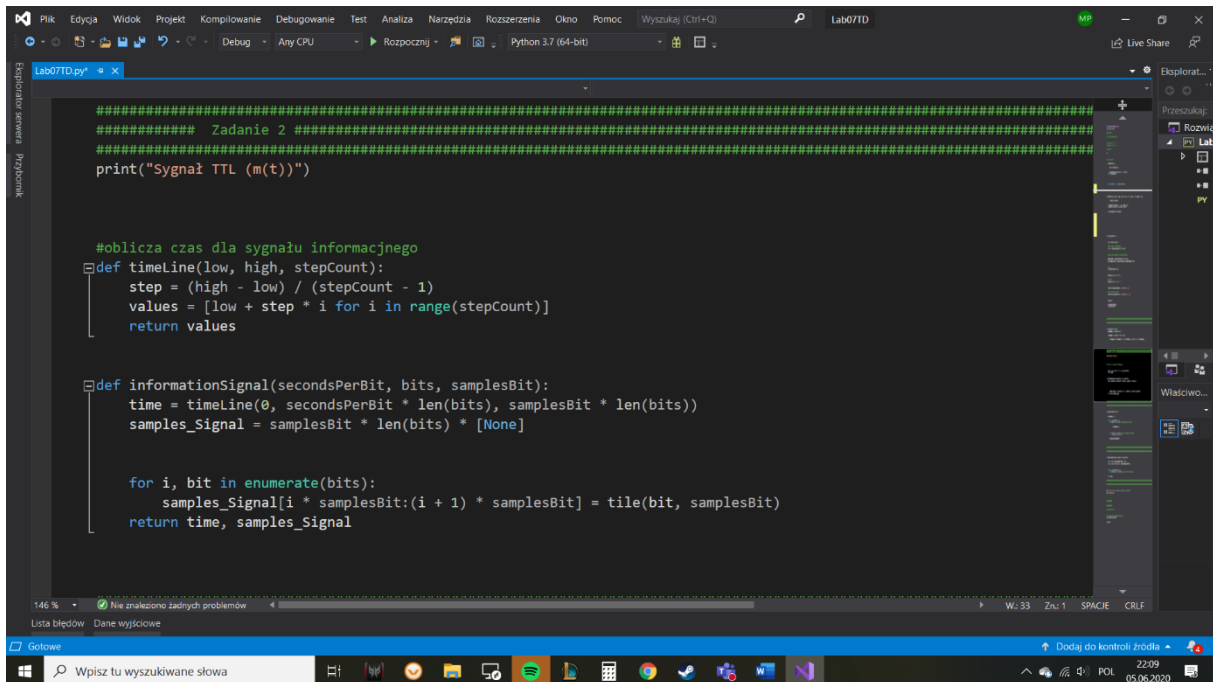


Funkcja rysująca wykresy.



Funkcja generująca sygnał zegarowy

2/.



```
##### Zadanie 2 #####
print("Sygnał TTL (m(t))")

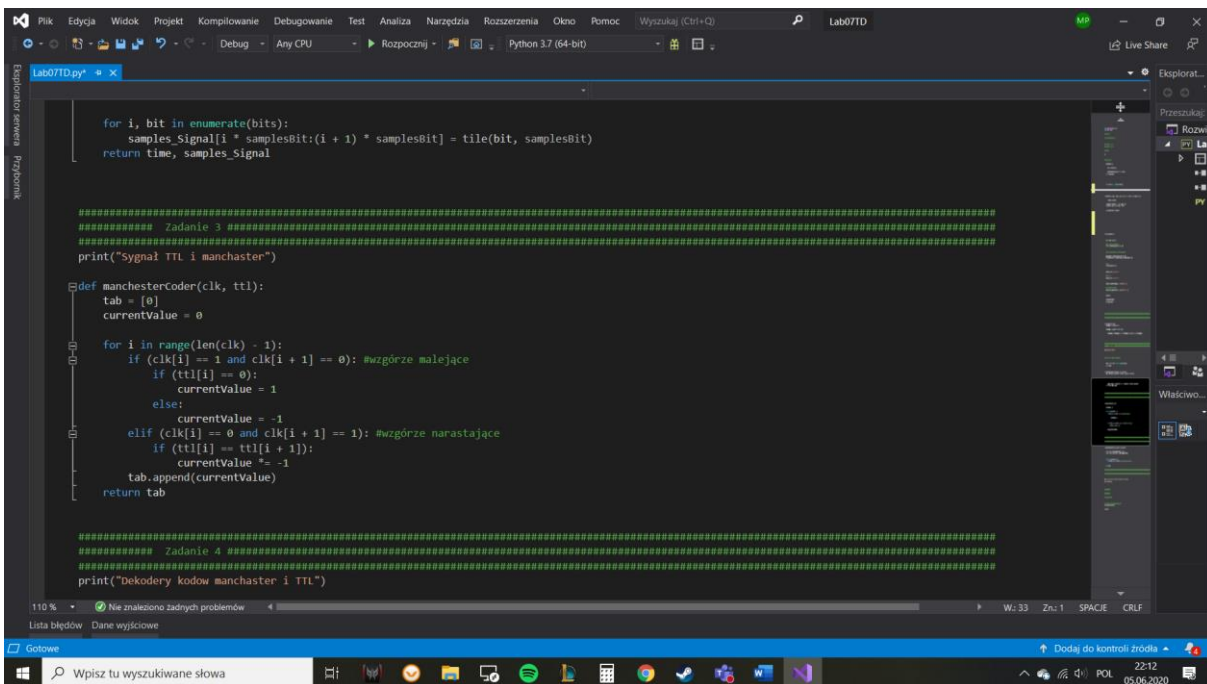
#oblicza czas dla sygnału informacyjnego
def timeline(low, high, stepCount):
    step = (high - low) / (stepCount - 1)
    values = [low + step * i for i in range(stepCount)]
    return values

def informationSignal(secondsPerBit, bits, samplesBit):
    time = timeline(0, secondsPerBit * len(bits), samplesBit * len(bits))
    samples_Signal = samplesBit * len(bits) * [None]

    for i, bit in enumerate(bits):
        samples_Signal[i * samplesBit:(i + 1) * samplesBit] = tile(bit, samplesBit)
    return time, samples_Signal
```

Funkcja dla sygnału informacyjnego

3/.



```
for i, bit in enumerate(bits):
    samples_Signal[i * samplesBit:(i + 1) * samplesBit] = tile(bit, samplesBit)
return time, samples_Signal

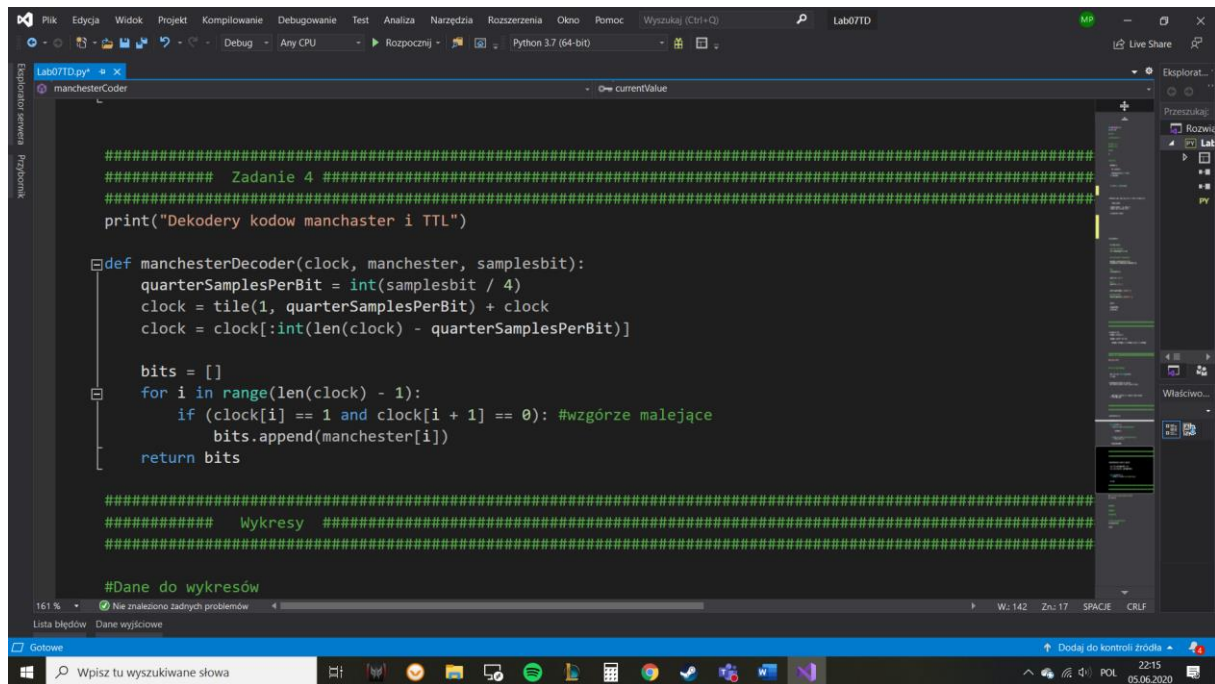
##### Zadanie 3 #####
print("Sygnał TTL i manchester")

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

##### Zadanie 4 #####
print("Dekodery kodów manchester i TTL")
```

Funkcja generująca przebieg sygnału Manchester

4/.



The screenshot shows a Python IDE with a file named 'Lab07TD.py'. The code is for a Manchester decoder. It starts with a comment 'Zadanie 4' and a print statement 'Dekodery kodow manchester i TTL'. The main function 'manchesterDecoder' takes 'clock', 'manchester', and 'samplesbit' as arguments. It calculates 'quarterSamplesPerBit' and creates a 'clock' array. It then iterates over the 'clock' array to find the Manchester bits. The code also includes comments for 'Wykresy' (Plots) and '#Dane do wykresow' (Data for plots).

```
##### Zadanie 4 #####
print("Dekodery kodow manchester i TTL")

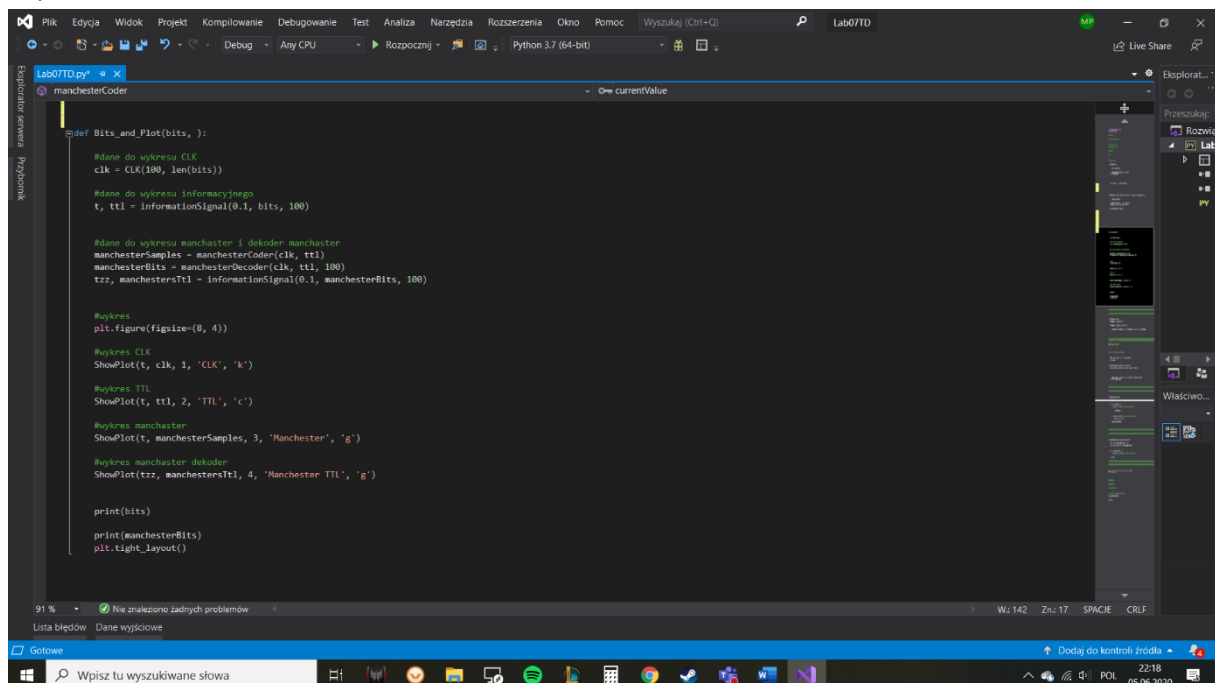
def manchesterDecoder(clock, manchester, samplesbit):
    quarterSamplesPerBit = int(samplesbit / 4)
    clock = tile(1, quarterSamplesPerBit) + clock
    clock = clock[:int(len(clock) - quarterSamplesPerBit)]

    bits = []
    for i in range(len(clock) - 1):
        if (clock[i] == 1 and clock[i + 1] == 0): #wzgórze malejące
            bits.append(manchester[i])
    return bits

##### Wykresy #####
##### Dane do wykresow #####
```

Dekoder kodu Manchester

Wyniki/.



The screenshot shows the same Python IDE with the 'Lab07TD.py' file. The code now includes a function 'def Bits_and_Plot(bits,):' which generates plots for the clock, information signal, and Manchester signal. It uses 'plt.figure' and 'plt.show' to display the plots. The code also includes comments for 'Wykresy' (Plots) and '#Dane do wykresow' (Data for plots).

```
def Bits_and_Plot(bits, ):
    #Dane do wykresu CLK
    clk = CLK(100, len(bits))

    #Dane do wykresu informacyjnego
    t, ttl = InformationSignal(0.1, bits, 100)

    #Dane do wykresu manchester i dekodera manchester
    manchesterSamples = manchesterCoder(clk, ttl)
    manchesterBits = manchesterDecoder(clk, ttl, 100)
    tzz, manchestersTtl = InformationSignal(0.1, manchesterBits, 100)

    #Wykres
    plt.figure(figsize=(8, 4))

    #Wykres CLK
    ShowPlot(t, clk, 1, 'CLK', 'k')

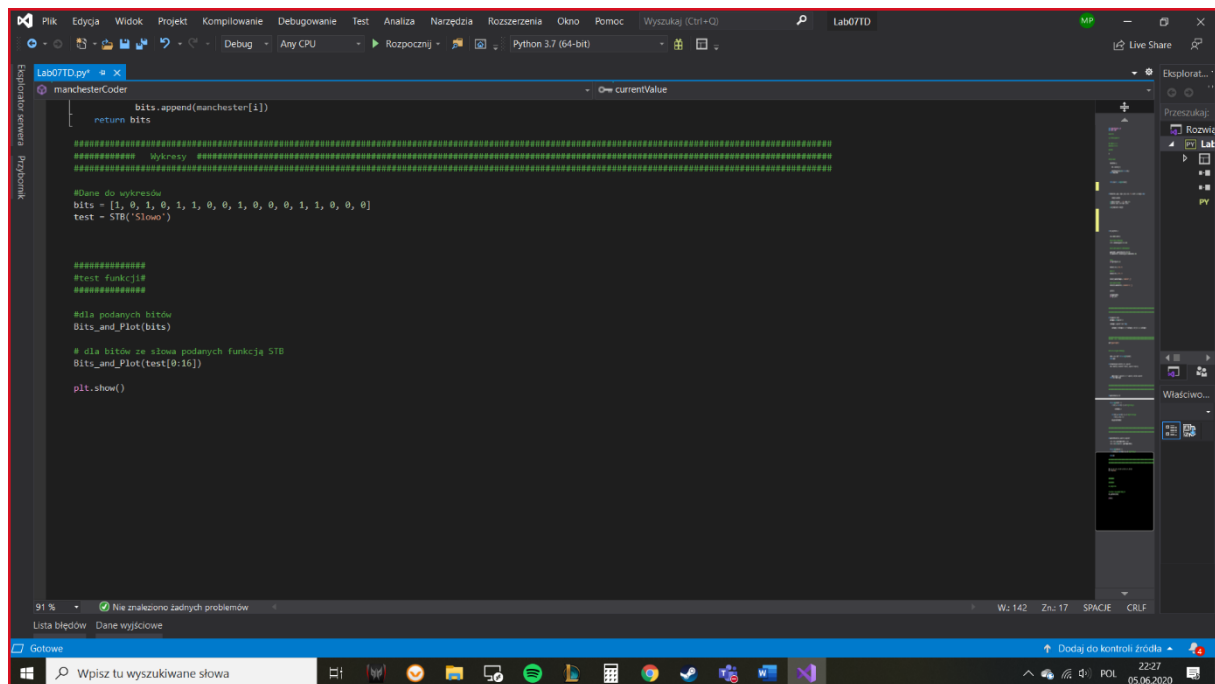
    #Wykres TTL
    ShowPlot(t, ttl, 2, 'TTL', 'c')

    #Wykres manchester
    ShowPlot(t, manchesterSamples, 3, 'Manchester', 'g')

    #Wykres manchester dekodera
    ShowPlot(tzz, manchestersTtl, 4, 'Manchester TTL', 'g')

    print(bits)
    print(manchesterBits)
    plt.tight_layout()
```

Funkcja generująca wykresy Clk, Sygnał Inf, TTL i Manchester

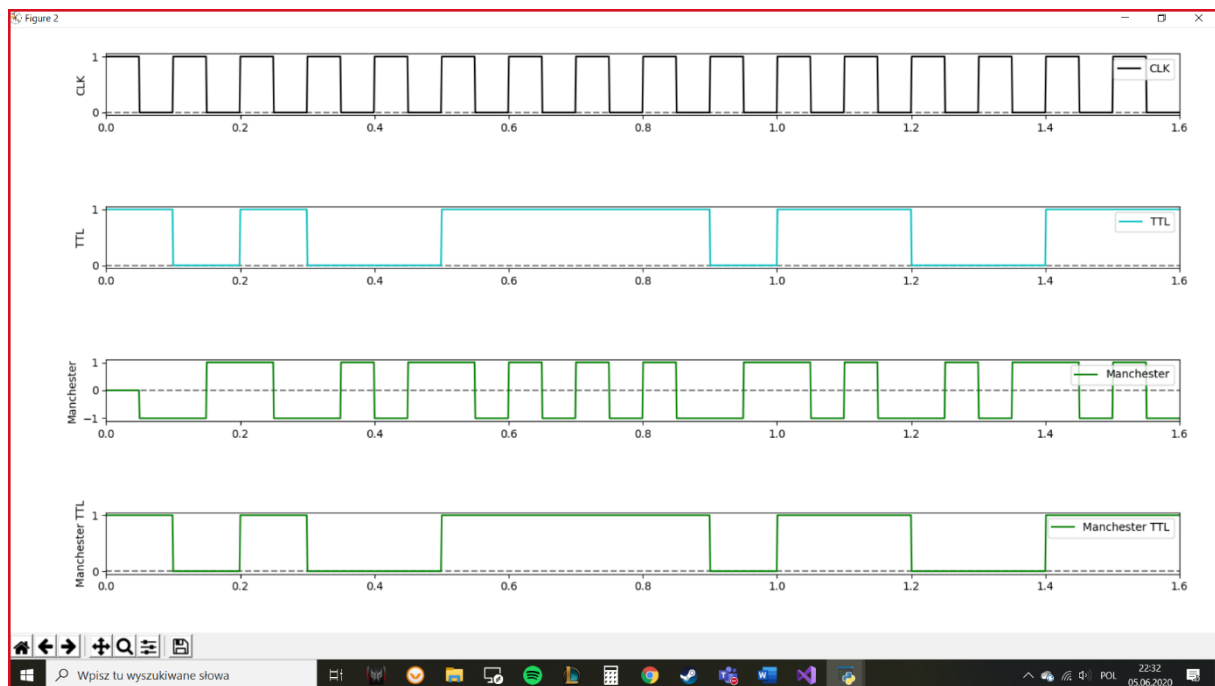


Testowanie poprawności działania przez podanie bitów oraz przez funkcję STB ze słowem „slowo”

Wykresy/.



Za pomocą podanych bitów.



Za pomocą funkcji STB.