import tkinter as tk

from tkinter import Frame, Button, Canvas

import matplotlib

matplotlib.use('TkAgg')

import numpy as np

import matplotlib.pyplot as plt

from math import pi

import math

import pickle

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

#from matplotlib.backends.backend\_tkagg import ( FigureCanvasTkAgg, NavigationToolbar2Tk)

from matplotlib.figure import Figure

from matplotlib.lines import Line2D

import matplotlib.animation as animation

import pylab

from scipy.signal import butter, lfilter

from scipy.signal import freqz

from scipy.ndimage import gaussian\_filter1d

import socket

import time

import datetime as dt

import random

numberconvert = 0

HEADERSIZE =100

numberlist = []

number\_list=[]

a=[]

type\_of\_measurement = ""

typeofelement=""

range2=0

list1=[]

input\_volt=0

def convert(x):

    c = (x >> 8) & 0xff

    f = x & 0xff

    return c, f

def butter\_bandpass(lowcut, highcut, fs, order=5):

    nyq = 0.5 \* fs

    low = lowcut / nyq

    high = highcut / nyq

    b, a = butter(order, [low, high], btype='band')

    return b, a

def butter\_bandpass\_filter(data, lowcut, highcut, fs, order=5):

    b, a = butter\_bandpass(lowcut, highcut, fs, order=order)

    y = lfilter(b, a, data)

    return y

def butter\_lowpass(lowcut, fs, order=5):

    nyq = 0.5 \* fs

    low = lowcut / nyq

    b, a = butter(order, [low], btype='low')

    return b, a

def butter\_lowpass\_filter(data, lowcut, fs, order=5):

    b, a = butter\_lowpass(lowcut, fs, order = order)

    y = lfilter(b, a, data)

    return y

def plot ():

    input\_volt=int(input\_volt\_amplitude.get())#----------input from amplitude

    fs=int(sampling.get())

    #range2=440

    #input\_volt=230#-------------input from amplitude

    type\_of\_measurement = measurement\_choices.get()

    typeofelement=str(type\_of\_measurement)

    range2 = choices.get()

    print("Type of measurement : ",type\_of\_measurement)

    print("Range is : ",range2)

    numberlist.append(type\_of\_measurement)

    numberlist.append(str(range2))

    numberlist.append(str(input\_volt))

    if type\_of\_measurement == "Voltage - AC":

        f=int(signal.get())

        lowcut = 40.0

        highcut = 70.0

        o = 3

        if input\_volt<range2:

            print("Signal Frequency is ",f)

            print("Input Voltage(Amplitude) is ",input\_volt)

            print("Sampling Frequency is ",fs)

            #print("Range is ",range2)

            #Parameters

            #input\_volt=230#-------------input from amplitude

            amp = 1.414\*input\_volt       #          (Amplitude)

            #f = 50        #      (Frequency)---------------signal frequency

            #fs = 5000    #     (Sample Rate)-----------------sampling freq

            T = 1/f

            Ts = 1/fs

            harmonic\_amp1=0.05\*amp

            harmonic\_amp2=0.05\*amp

            #attenuation\_factor=(amp+harmonic\_amp1+harmonic\_amp2)/2.5

            attenuation\_factor= (range2 + 0.5\*range2) /2.5

            #attenuation\_factor\_value\_label['text'] = attenuation\_factor

            numberlist.append(str(attenuation\_factor))

            attenuation.set(attenuation\_factor)

            continous  = True

            #f = Figure(figsize=(6,4), dpi=100)

            fig = plt.Figure()

            x = np.arange(fs)

            #print(x)

            yout= [ ((amp\*np.sin(2\*np.pi\*f \* (i/fs)))+(0.05\*amp\*np.sin(6\*pi\*f \* (i/fs)))+(0.05\*amp\*np.sin(12\*pi\*f \* (i/fs)))) for i in x ]

            yo= [ ((amp\*np.sin(2\*np.pi\*f \* (i/fs)))+(0.05\*amp\*np.sin(6\*pi\*f \* (i/fs)))+(0.05\*amp\*np.sin(12\*pi\*f \* (i/fs))))\*(1/attenuation\_factor) for i in x ]

                    #Add If condition for AC selection of configurator

            adc=[((yo[i]/5)\*65536)+(32768) for i in x]

            for i in x:

                #print((y[i]))

                op1 = 32768 + butter\_bandpass\_filter(adc, lowcut, highcut, fs, order=o)

                number\_list = op1.tolist()

                #op2 = gaussian\_filter1d(op1, 4)

                #print(math.ceil(op2[i]))

                #adc=int((y[i]/5)\*65536)+(32768)

                #print(adc)

                    #End of IF condition

                    #Add ELSE IF condition for DC selection of configurator

                    #End ELSE IF condition

            #adc=((y/10)\*65536)+(32768)

            print(list1)

            #print(type(op1))

            for i in number\_list:

                numberlist.append(convert(math.ceil(i)))

            #print(numberlist)

            class aScope(object):

                def \_\_init\_\_(self, ax, maxt=2\*T, dt=Ts):

                    self.ax = ax

                    self.dt = dt

                    self.maxt = maxt

                    self.tdata = [0]

                    self.ydata = [0]

                    self.line = Line2D(self.tdata, self.ydata)

                    self.ax.add\_line(self.line)

                    self.ax.set\_ylim(-350,350)

                    self.ax.set\_xlim(0, self.maxt)

                def aupdate(self, y):

                    lastt = self.tdata[-1]

                    if continous :

                        if lastt > self.tdata[0] + self.maxt:

                            self.ax.set\_xlim(lastt-self.maxt, lastt)

                    t = self.tdata[-1] + self.dt

                    self.tdata.append(t)

                    self.ydata.append(y)

                    self.line.set\_data(self.tdata, self.ydata)

                    return self.line,

            def sineEmitter():

                for i in x:

                    yield math.ceil(yout[i])

            #fig, ax = plt.subplots()

            #f = Figure(figsize=(6,4), dpi=100)

            fig = plt.Figure(figsize=(6,4))

            canvas = FigureCanvasTkAgg(fig, master=root)

            canvas.get\_tk\_widget().place(x=0, y=190)

            ax1 = fig.add\_subplot(111)

            ax1.set\_xlabel("Time")

            ax1.set\_ylabel("Amplitude")

            line, = ax1.plot(x, np.sin(x))

            scope = aScope(ax1)

            # pass a generator in "sineEmitter" to produce data for the update func

            plot.ani = animation.FuncAnimation(fig, scope.aupdate, sineEmitter, interval=10,blit=True)

        else:

            #error messege

            tk.messagebox.showerror("Error", "Exceeds the range")

            tk.messagebox.showinfo("Hint","The Input Voltage(Amplitude) is always lesser than range value.")

    elif type\_of\_measurement == "Voltage - DC":

        #fs = 5000

        lowcut = 25

        o = 3

        N = 40000

        N2 = 40000

        Ts=1/fs

        if input\_volt<range2:

            print("Input voltage(Amplitude) is ",input\_volt)

            print("Sampling Frequency is ",fs)

            #print("Range is ",range2)

            continous  = True

            #time = np.arange(0\*np.pi, 5\*np.pi, 0.01)

            #amplitude1 = 100\*np.sin(time)

            #deactivate the signal frequency

            input\_volt=int(input\_volt\_amplitude.get())#----------input from amplitude

            #ip = 100 + ((3\*np.random.randn(N)))

            attenuation\_factor = range2/5

            attenuation.set(attenuation\_factor)

            yout = (input\_volt + ((0.01\*input\_volt\*np.random.randn(N))))

            ip = (input\_volt + ((0.01\*input\_volt\*np.random.randn(N))))\*5/range2

            adc=((ip/5)\*65536)

            '''

            i = 0

            it = 0

            while(it < N):

            #while(True):

                while(i < N2):

                    #print (ip[i])

                    #print ("    ")

                    op = butter\_lowpass\_filter(adc,lowcut,fs,order = o)

                    op1 = gaussian\_filter1d(op, 12)

                    #print(op1[i])

                    numberlist.append(convert(math.ceil(op1[i])))

                    #print (math.ceil(op2[i]))

                    i = i + 1

                it = it + N2

            plt.plot(ip, 'k', label='original data')

            plt.plot(op1, '--', label='filtered')

            plt.legend()

            plt.grid()

            plt.show()

            #f = Figure(figsize=(6,4), dpi=100)

            fig = plt.Figure()

            x = np.arange(fs)

            #print(numberlist)

            #print(x)

            '''

            x = []

            #input\_volt = 50

            N = fs

            y = []

            def animate(i,x,y):

                x.append(i)

                y.append(input\_volt + ((0.01\*input\_volt\*random.randint(0,10))))

                i+=1

                #x=x[-50:500]

                #y=y[-50:500]

                x=x[:fs]

                y=y[:fs]

                ax1.clear()

                ax1.plot(x,y)

                plt.xticks(rotation=45, ha='right')

            fig = plt.Figure(figsize=(6,4))

            canvas = FigureCanvasTkAgg(fig, master=root)

            canvas.get\_tk\_widget().place(x=0, y=190)

            plot.ani = animation.FuncAnimation(fig, animate, fargs=(x,y),interval=1)

            ax1 = fig.add\_subplot(111)

            ax1.set\_xlabel("Time")

            ax1.set\_ylabel("Amplitude")

            line, = ax1.plot(x, np.sin(x))

        else:

            #error messege

            tk.messagebox.showerror("Error", "Exceeds the range")

            tk.messagebox.showinfo("Hint","The Amplitude value is always lesser than range value.")

def reset():

    signal.set(0)

    input\_volt\_amplitude.set(0)

    sampling.set(0)

    measurement\_choices.set("Voltage - AC")

    choices.set(0)

    attenuation.set(0)

    #canvas.delete('all')

    #animation.destroy()

def on\_option\_change(event):

    selected = measurement\_choices.get()

    attenuation.set(0)

    if selected == "Voltage - AC":

        input\_range = {110,230,440,500}

    elif selected == "Voltage - DC":

        input\_range = {10,24,48}

    input\_range=sorted(input\_range)

    range\_label= tk.Label(root,text="",bg='White',fg='black',font='Helvetica 8 bold')

    range\_option = tk.OptionMenu(root,choices, \*input\_range)

    choices.set(0)

    range\_option.config(bg = "LightYellow2")

    range\_option.configure(width=15)

    if selected == "Voltage - AC":

        signal.set(0)

        input\_volt\_amplitude.set(0)

        sampling.set(0)

        range\_label['text'] = "Range"

        range\_label.place(x=300, y=10)

        #choices.set(0)

        range\_option.configure(width=15)

        range\_option.place(x=260, y=35)

        #signal\_freq\_label.place(x=30, y=100)

        #signal\_freq\_entry.place(x=30, y=130)

        signal\_freq\_label.place(x=430, y=100)

        signal\_freq\_entry.place(x=430, y=130)

    elif selected == "Voltage - DC":

        signal.set(0)

        input\_volt\_amplitude.set(0)

        sampling.set(0)

        range\_label['text'] = "Range"

        range\_label.place(x=300, y=10)

        #choices.set(0)

        range\_option.configure(width=15)

        range\_option.place(x=260, y=35)

        signal\_freq\_label.place\_forget()

        signal\_freq\_entry.place\_forget()

#Execution starts here

root = tk.Tk()

root.title("Configurator")

title\_bar = Frame(root, bg='brown', relief='raised', bd=2)

title\_bar.pack()

root.geometry("600x700")

root.configure(bg='#FFFFFF')

#Variable data types

measurement\_choices = tk.StringVar()

ac\_choices = tk.IntVar()

dc\_choices = tk.IntVar()

choices=tk.IntVar()

#Assigning values

measurement\_type = {"Voltage - AC", "Voltage - DC"}

#Label - Type of Measurement

type\_label=tk.Label(root,text="Type of Measurement",bg='White',fg='black',font='Helvetica 8 bold')

type\_label.place(x=50, y=10)

measurement\_choices.set("Voltage - AC")

#Option Menu to select Measurement type

measur\_type = tk.OptionMenu(root,measurement\_choices, \*measurement\_type, command=on\_option\_change)

measur\_type.config(bg = "LightYellow2")

measur\_type.configure(width=15)

measur\_type.place(x=50, y=35)

#Signal Frequency

signal\_freq\_label = tk.Label(text="Signal Frequency",bg='White',fg='black',font='Helvetica 8 bold')

#signal\_freq\_label.place(x=30, y=100)

signal\_freq\_label.place(x=430, y=100)

signal = tk.DoubleVar()

signal\_freq\_entry = tk.Entry(root, width = 15, textvariable = signal)

#signal\_freq\_entry.place(x=30, y=130)

signal\_freq\_entry.place(x=430, y=130)

#Amplitude

amplitude\_label = tk.Label(text="Input Voltage",bg='White',fg='black',font='Helvetica 8 bold')

amplitude\_label.place(x=160, y=100)

input\_volt\_amplitude = tk.DoubleVar()

amplitude\_entry = tk.Entry(root, width = 15, textvariable = input\_volt\_amplitude)

amplitude\_entry.place(x=160, y=130)

#Sampling Frequency

sampling\_freq\_label = tk.Label(text="Sampling Frequency",bg='White',fg='black',font='Helvetica 8 bold')

sampling\_freq\_label.place(x=290, y=100)

sampling = tk.DoubleVar()

sampling\_freq\_entry = tk.Entry(root, width = 15, textvariable = sampling)

sampling\_freq\_entry.place(x=290, y=130)

#Attenuation Factor

attenuation\_factor\_label = tk.Label(text="Attenuation Factor",bg='White',fg='black',font='Helvetica 8 bold',width=15)

#attenuation\_factor\_label.place(x=430, y=100)

attenuation\_factor\_label.place(x=30, y=100)

attenuation = tk.DoubleVar()

attenuation\_factor\_value\_label = tk.Label(text="Attenuation Value",bg='DarkSeaGreen3',fg='black',font='Helvetica 8 bold',width=15, textvariable = attenuation)

#attenuation\_factor\_value\_label.place(x=430, y=130)

attenuation\_factor\_value\_label.place(x=30, y=130)

#Draw button

graph\_button = tk.Button(root, text="Execute",bg='brown',fg='white', command=plot)

graph\_button.place(x=45, y=160)

#Reset button

#reset\_button = tk.Button(root,text="Reset", bg='brown',fg='white', command=reset)

#reset\_button.place(x=250, y=160)

root.mainloop()

HOST = socket.gethostbyname(socket.gethostname())

PORT = 7874

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

    s.bind((HOST, PORT))

    s.listen()

    conn, addr = s.accept()

    with conn:

        print('Connected by', addr)

       # data = type\_of\_measurement

        #conn.send(data.encode())

        while True:

            for i in numberlist :

                for j in i :

                    if not i:

                        break

                    data=str(numberlist)

                    data=data.encode()

                    conn.sendall(data)

conn.close()

import socket

import numpy as np

import matplotlib.pyplot as plt

import math

from scipy.ndimage import gaussian\_filter1d

import pickle

import tkinter as tk

from tkinter import Frame, Button, Canvas

from matplotlib.lines import Line2D

import matplotlib.animation as animation

from math import pi

from scipy.signal import butter, lfilter

from scipy.signal import freqz

from numpy.compat.py3k import long

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

def plot():

    iterDC = 0

    iterAC = 0

    maxval = len(arr)

    x = np.arange(maxval, step = 1)

    if (type\_of\_measurement == 'Voltage - DC'):

        while(iterDC < maxval ):

            opc.append((lis[iterDC]/65536)\*Range)

            iterDC = iterDC+1

    elif (type\_of\_measurement == 'Voltage - AC'):

        while(iterAC < maxval ):

            opc.append(((lis[iterAC]-32768)/65536)\*(Range)\*2.5\*1.26)

            iterAC = iterAC+1

    InVolt\_label\_value1.set(input\_volt)

    range\_label\_value1.set(Range)

    mt1['text']=type\_of\_measurement

    u1['text']="Volts"

    attenuation\_value.set(attenuation\_factor)

    fig = plt.Figure(figsize=(6,5))

    #plt.bar(x=x, height=y)

    plt.xticks(x, rotation=90)

    canvas = FigureCanvasTkAgg(fig, master=root)

    canvas.draw()

    canvas.get\_tk\_widget().place(x=0, y=200)

    ax1 = fig.add\_subplot(111)

    ax1.set\_xlabel("Time")

    ax1.set\_ylabel("Amplitude")

    line, = ax1.plot(x, np.sin(x))

def reset():

    InVolt\_label\_value1.set(0)

    u1['text']=" "

    mt1['text']=" "

    range\_label\_value1.set(0)

    attenuation\_value.set(0)

def exitWindow():

    root.destroy()

HOST = socket.gethostbyname(socket.gethostname())

PORT = 7874

HEADERSIZE = 10

lis= []

lis2=[]

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

    s.connect((HOST, PORT))

    data = s.recv(640016)

    data = data.decode('utf-8')

    data = eval(data)

    type\_of\_measurement = data[0]

    Range = int(data[1])

    input\_volt=float(data[2])

    attenuation\_factor=float(data[3])

    print(f"type\_of\_measurement:{type\_of\_measurement}")

    print(f"Range:{Range}")

    print(f"attenuation\_factor:{attenuation\_factor}")

    print(f"input\_volt:{input\_volt}")

    #opc = []

    del data[0:4]

    for i in data :

        if not i:

            break

        j=(i[0]<<8)+i[1]

        lis.append(j)

    opc = []

    arr = np.array(lis)

    #print(arr)

    iterDC = 0

    iterAC = 0

    maxval = len(arr)

    x = np.arange(maxval, step = 1)

    '''

    if (type\_of\_measurement == 'Voltage - DC'):

        while(iterDC < maxval ):

            opc.append((lis[iterDC]/65536)\*Range)

            iterDC = iterDC+1

    if (type\_of\_measurement == 'Voltage - AC'):

        while(iterAC < maxval ):

            opc.append(((lis[iterAC]-32768)/65536)\*(Range)\*2.5\*1.26)

            iterAC = iterAC+1

    '''

    #opiter = 0

    #opc\_list = opc.tolist()

    #for opiter in x:

        #print(opc\_list[opiter])

        #opiter = opiter + 1

    #plt.plot(x, opc, color='green', linestyle='dashed', linewidth = 3, marker='o', markerfacecolor='blue', markersize=5)

    #plt.grid()

    #plt.show()

#Execution starts here

root = tk.Tk()

root.title("Display")

title\_bar = Frame(root, bg='brown', relief='raised', bd=2)

title\_bar.pack()

root.geometry("600x800")

root.configure(bg='#FFFFFF')

#Variable data types

InVolt\_label\_value1 = tk.DoubleVar()

unit\_label\_value1 = tk.StringVar()

mtype\_label\_value1 = tk.StringVar()

range\_label\_value1 = tk.DoubleVar()

attenuation\_value = tk.DoubleVar()

#Label - Measurement Calculation

iv=tk.Label(root,text="Input Voltage",bg='White',fg='black',font='Helvetica 8 bold')

iv.place(x=35, y=20)

#Measurement Calculation value

mCalc\_label\_value = tk.Label(text="",bg='DarkSeaGreen3',fg='black',font='Helvetica 8 bold',width=15, textvariable = InVolt\_label\_value1)

mCalc\_label\_value.place(x=50, y=40)

#Unit

u = tk.Label(text="Unit",bg='White',fg='black',font='Helvetica 8 bold')

u.place(x=200,y=20)

#Unit value

u1 = tk.Label(text="",bg='DarkSeaGreen3',fg='black',font='Helvetica 8 bold',width=15)

u1.place(x=200, y=40)

#Measurement Type

mt = tk.Label(text="Measurement Type",bg='White',fg='black',font='Helvetica 8 bold',width=15)

mt.place(x=50, y=90)

#Measurement Type value

mt1 = tk.Label(text="",bg='DarkSeaGreen3',fg='black',font='Helvetica 8 bold',width=15)

mt1.place(x=50, y=120)

#Range

r = tk.Label(text="Range",bg='White',fg='black',font='Helvetica 8 bold',width=15)

r.place(x=200, y=90)

#Range value

r1 = tk.Label(text="",bg='DarkSeaGreen3',fg='black',font='Helvetica 8 bold',width=15, textvariable = range\_label\_value1)

r1.place(x=200, y=120)

#Range

a = tk.Label(text="Attenuation factor",bg='White',fg='black',font='Helvetica 8 bold',width=15)

a.place(x=360, y=90)

#Range value

a1 = tk.Label(text="",bg='DarkSeaGreen3',fg='black',font='Helvetica 8 bold',width=15, textvariable = attenuation\_value)

a1.place(x=355, y=120)

#Draw button

graph\_button = tk.Button(root, text="Execute",bg='brown',fg='white',width=10, command=plot)

graph\_button.place(x=50, y=170)

#Reset button

reset\_button = tk.Button(root,text="Reset", bg='brown',fg='white',width=10, command=reset)

reset\_button.place(x=400, y=170)

#reset\_button.place(x=225, y=170)

'''

#Exit button

exit\_button = tk.Button(root,text="Exit", bg='brown',fg='white',width=10, command=exitWindow)

exit\_button.place(x=400, y=170)

'''

root.mainloop()