Chapter 7 - Microwave Circuit Simulator

Design, Features, and Specifications

RF & Microwave circuit design is a specialized field of electrical engineering for frequencies from 300 MHz to 100 GHz. This advanced project will address the design and development of a RF/Microwave Circuit Simulator. Note that analysis for high frequency circuits uses S-Parameters rather than Voltage and Current.

Program Design & Architecture:

- Modify the Digital Simulator to create the Microwave Simulator
- Object-Oriented Programming (OOP) - class abstraction, inheritance, & polymorphism
- KISS - Keep it simple, silly
- DRY - Don't repeat yourself
- SOC - Separation of concerns
- Python Modules & Packages
- User-interface
  - TopFrame class
  - File menu frame
  - Settings menu frame
  - Rotation button
  - Help menu frame
  - Left Frame Class with Circuit Component Menus
  - Canvas class
  - Mouse class
- Circuit components
  - Lumped Components
    - Resistor
    - Capacitor
    - Inductor
  - Ideal Components
    - Input Port
    - Output Port
  - Wires

- Straight Wire Class
- Segment Wire Class
- Elbow Wire Class
- Grid class

- Analysis
  - Linear S-Parameter Analysis
  - Raw data display
  - Line graphs
  - Smith chart

Key Technologies Needed:

- Microwave analysis library - Scikit-RF

Project Setup

Language: Python 3.11
IDE: PyCharm 2023.2.1 (Community Edition)
Project directory: D:/EETools/MicrowaveSimulator
Graphics library: CustomTkinter (https://customtkinter.tomschimansky.com/)

External libraries:

- pip install customtkinter
- python.exe -m pip install --upgrade pip
- pip install ctkcolorpicker
- pip install tkinter-tooltip
- pip install pyInstaller - Create .exe file
- pip install `jsonpickle`
- pip install matplotlib
- Add images and icons directories to the project.

RF/Microwave Analysis Library

- scikit-rf
- pip install scikit-rf

Scikit-RF Evaluation

- Scikit-RF Home Page
- Scikit-RF Documentation
- Github Repository

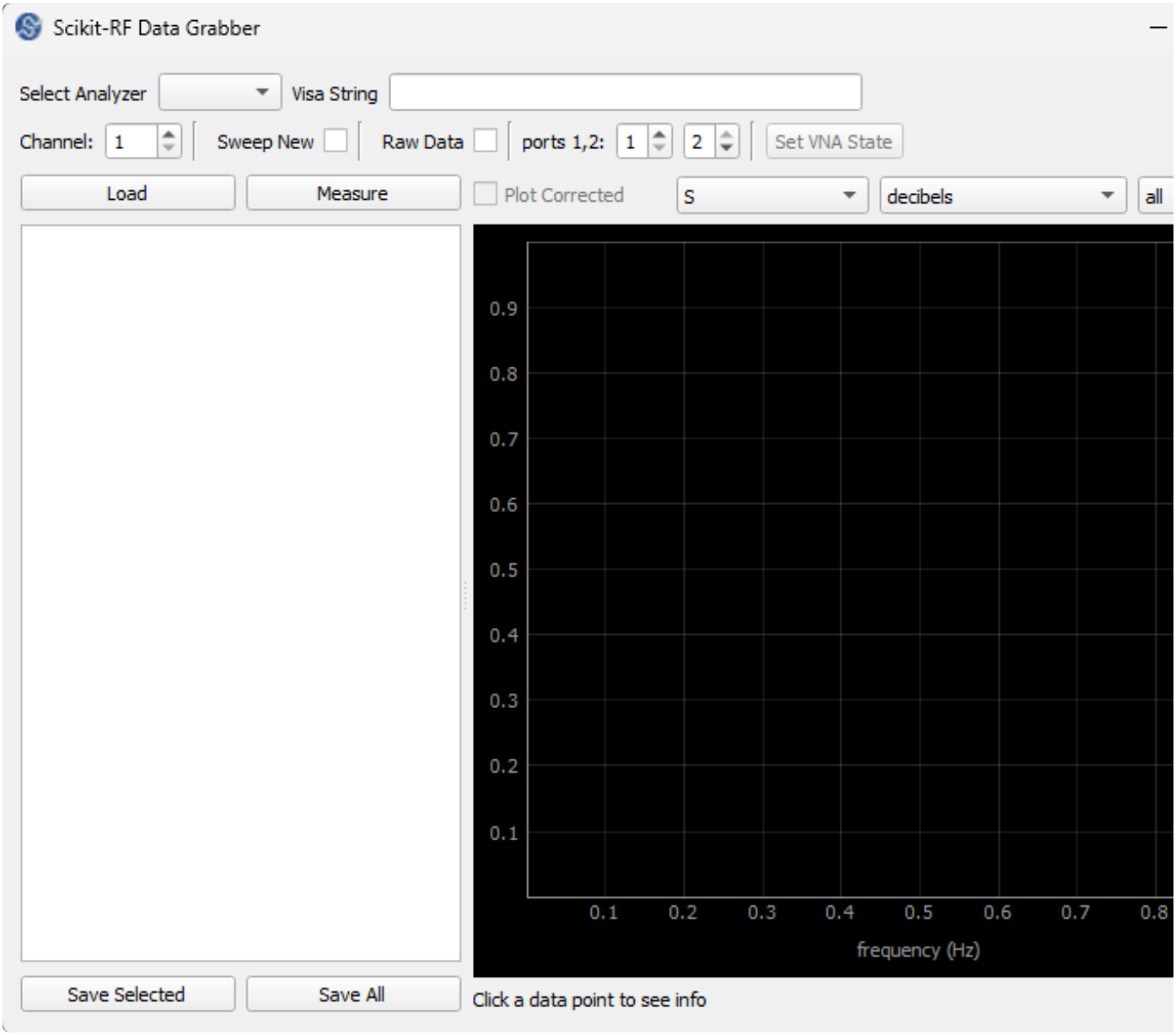
Features

- Microwave Network Operations:
  - Read/Write touchstone (.s2p, s?p) files
  - Arithmetic operations on scattering parameters
  - Cascade/De-embed 2-port networks
  - Frequency and port slicing and concatenation.
  - Connect n-port networks
  - s/z/y/abcd/t - parameter conversion
- Sets of Networks:
  - Statistical properties of NetworkSets
  - Methods to sort and visualize set behavior
- GUI support through qtapps
  - modular, re-useable apps
  - supports data retrieval, plotting, calibration and more.
- Plotting abilities:
  - Rectangular Plots (dB, mag, Phase, group delay)
  - Smith Chart
  - Automated Uncertainty bounds
- Offline Calibration:
  - One-Port: SOL, Least Squares, SDDL
  - Two-Port: TRL, Multiline TRL, SOLT, Unknown Thru, 8/16-Term
  - Partial : Enhanced Response, One-Port Two-Path
- Virtual Instruments (completeness varies by model)
  - VNAs: PNA, PNAX, ZVA, HP8510, HP8720
  - SA: HP8500
  - Others: ESP300
- Transmission Line Physics:
  - Coax, CPW, Freespace, Rectangular Waveguide, Distributed Circuit

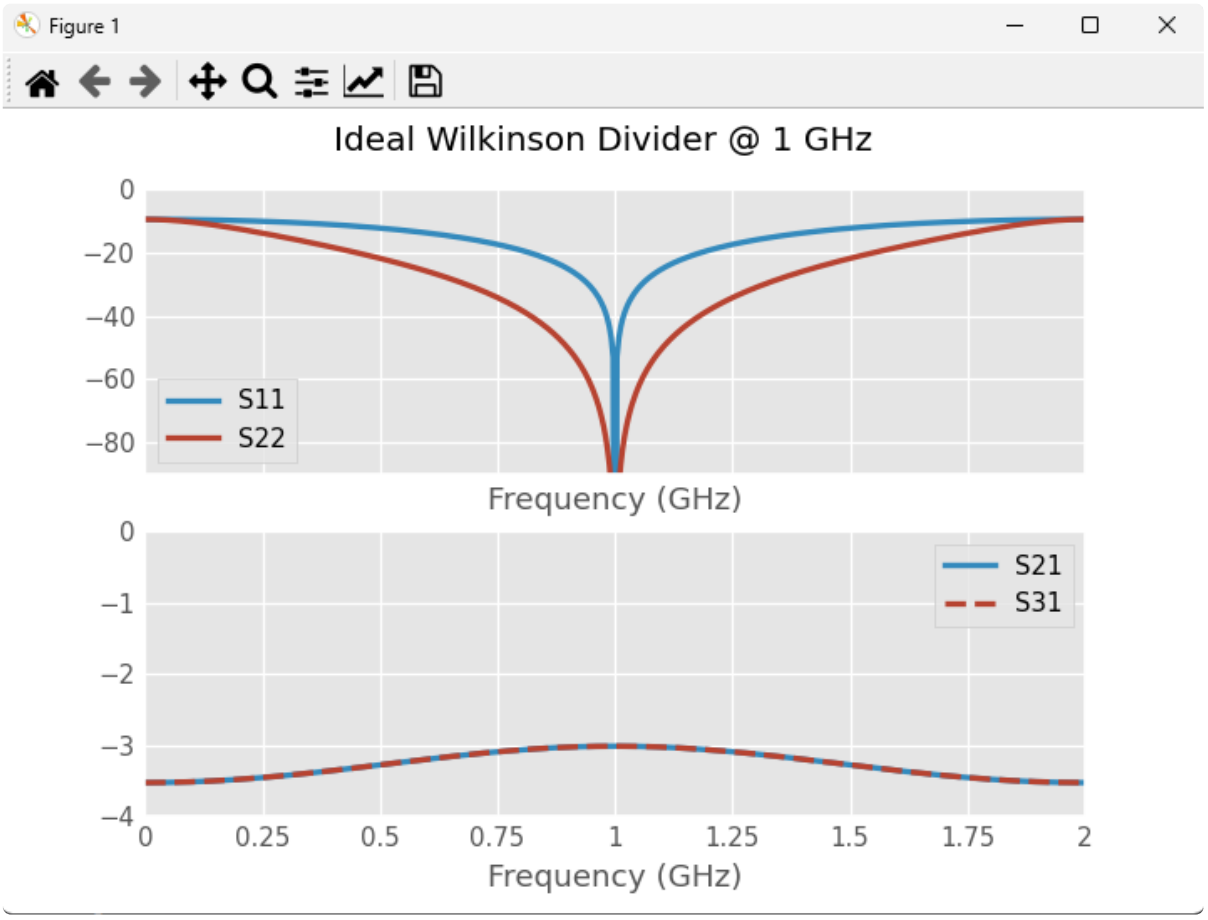
Media Classes

`DefinedGammaZ0` ([frequency, z0_port, z0, Z0, ...])	A media directly defined by its propagation constant and characteristic impedance.
`DistributedCircuit` ([frequency, z0_port, ...])	A transmission line mode defined in terms of distributed impedance and admittance values.
`RectangularWaveguide` ([frequency, z0_port, ...])	A single mode of a homogeneously filled rectangular waveguide.
`CircularWaveguide` ([frequency, z0_port, ...])	A single mode of a homogeneously filled Circular Waveguide
`CPW` ([frequency, z0_port, z0_override, z0, ...])	Coplanar waveguide.
`Coaxial` ([frequency, z0_port, z0_override, ...])	A coaxial transmission line defined in terms of its inner/outer diameters and permittivity.
`MLine` ([frequency, z0_port, z0_override, z0, ...])	A microstripline transmission line defined in terms of width, thickness and height on a given relative permittivity substrate.
`Freespace` ([frequency, z0_port, z0_override, ...])	A plane-wave (TEM Mode) in Freespace.
`DefinedAEpTandZ0` ([frequency, A, f_A, ep_r, ...])	Transmission line medium defined by A, Ep, Tand and Z0.

Example of a Scikit-rf GUI application



[Wilkinson Power Divider Example]

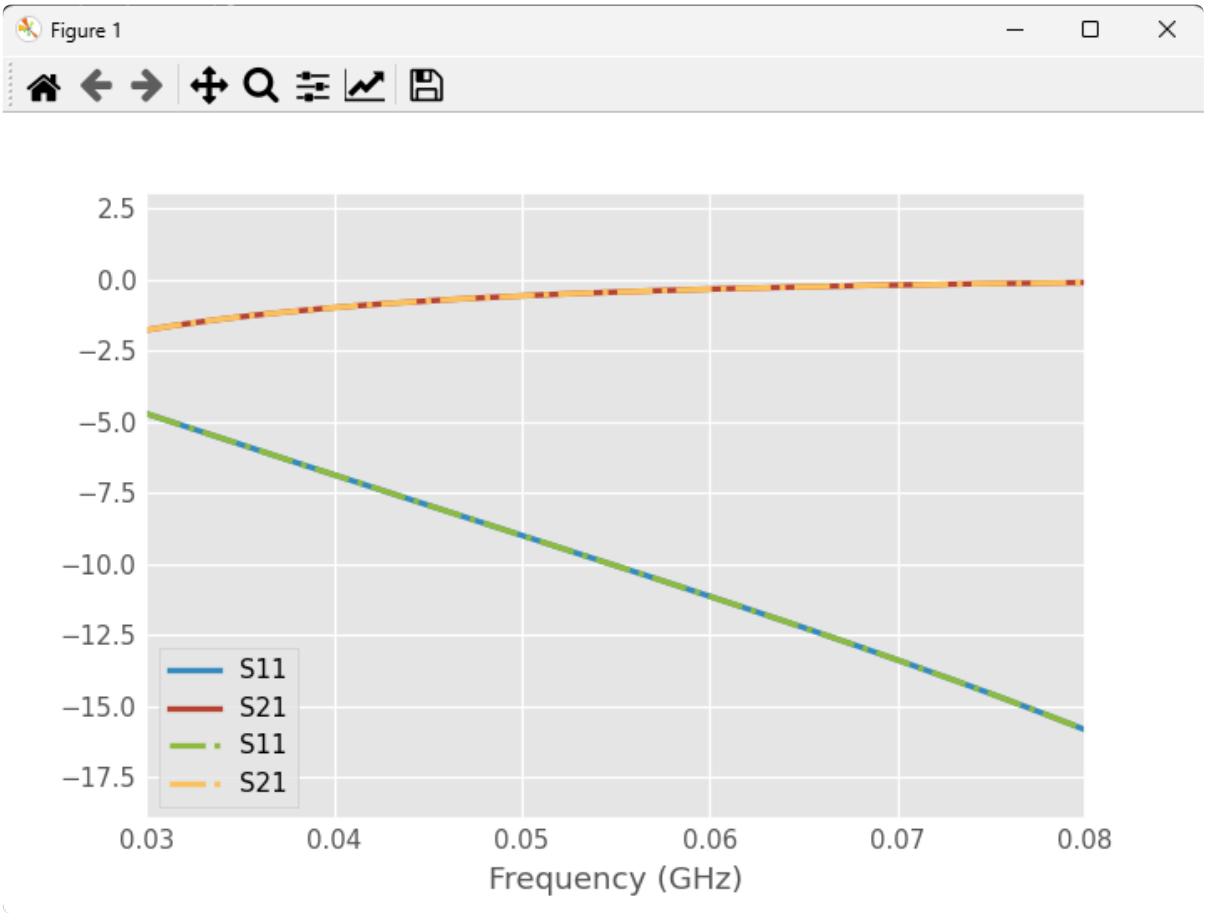


Sandbox/wilkinson_example.py

```
import numpy as np
import matplotlib.pyplot as plt
import skrf as rf

rf.stylely()

# frequency band
freq = rf.Frequency(start=0, stop=2, npoints=501, unit='GHz')

# characteristic impedance of the ports
z0_ports = 50

# resistor
R = 100
line_resistor = rf.media.DefinedGammaZ0(frequency=freq, z0=R)
resistor = line_resistor.resistor(R, name='resistor')

# branches
z0_branches = np.sqrt(2)*z0_ports
beta = freq.w/rf.c
```

```
line_branches = rf.media.DefinedGammaZ0(frequency=freq, z0=z0_branches,
gamma=0+beta*1j)

d = line_branches.theta_2_d(90, deg=True)  # @ 90°(lambda/4)@ 1 GHz is ~ 75 mm
branch1 = line_branches.line(d, unit='m', name='branch1')
branch2 = line_branches.line(d, unit='m', name='branch2')

# ports
port1 = rf.Circuit.Port(freq, name='port1', z0=50)
port2 = rf.Circuit.Port(freq, name='port2', z0=50)
port3 = rf.Circuit.Port(freq, name='port3', z0=50)

# Connection setup
# Note that the order of appearance of the port in the setup is important
connections = [
          [(port1, 0), (branch1, 0), (branch2, 0)],
          [(port2, 0), (branch1, 1), (resistor, 0)],
          [(port3, 0), (branch2, 1), (resistor, 1)]
        ]

# Building the circuit
C = rf.Circuit(connections)

fig, (ax1,ax2) = plt.subplots(2, 1, sharex=True)
C.network.plot_s_db(ax=ax1, m=0, n=0,  lw=2)  # S11
C.network.plot_s_db(ax=ax1, m=1, n=1,  lw=2)  # S22
ax1.set_ylim(-90, 0)
C.network.plot_s_db(ax=ax2, m=1, n=0,  lw=2)  # S21
C.network.plot_s_db(ax=ax2, m=2, n=0,  ls='--', lw=2)  # S31
ax2.set_ylim(-4, 0)
fig.suptitle('Ideal Wilkinson Divider @ 1 GHz')
plt.show()
```

Lumped Element Example

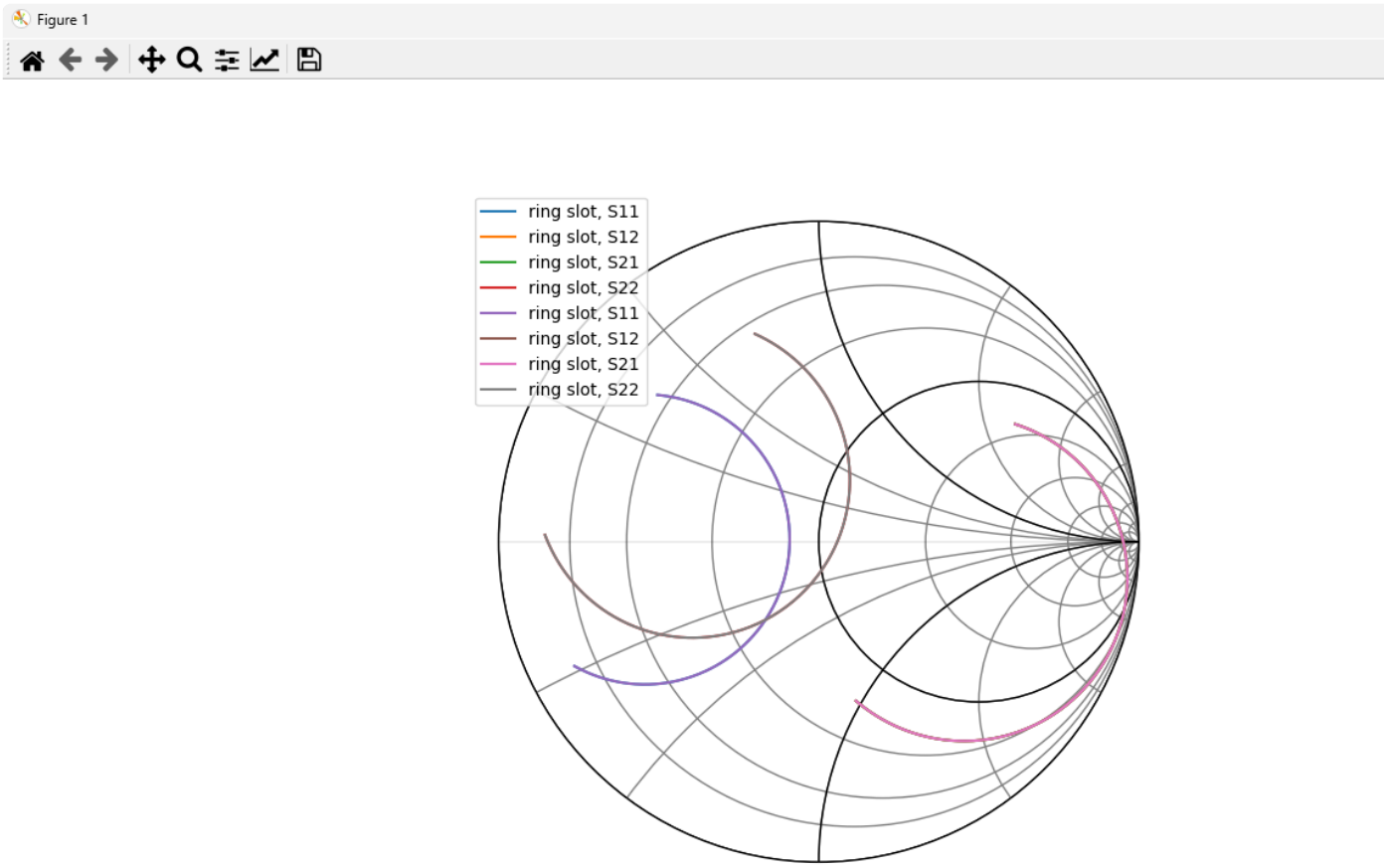


Sandbox/lumped_example.py

```
import numpy as np  # for np.allclose() to check that S-params are similar
import skrf as rf
import matplotlib.pyplot as plt
rf.stylely()

# reference LC circuit made in Designer
LC_designer = rf.Network('data/designer_capacitor_30_80MHz_simple.s2p')

# scikit-rf: manually connecting networks
line = rf.media.DefinedGammaZ0(frequency=LC_designer.frequency, z0=50)
LC_manual = line.inductor(24e-9) ** line.capacitor(70e-12)

# scikit-rf: using Circuit builder
port1 = rf.Circuit.Port(frequency=LC_designer.frequency, name='port1', z0=50)
port2 = rf.Circuit.Port(frequency=LC_designer.frequency, name='port2', z0=50)
cap = rf.Circuit.SeriesImpedance(frequency=LC_designer.frequency, name='cap',
z0=50,
                                 Z=1/(1j*LC_designer.frequency.w*70e-12))
ind = rf.Circuit.SeriesImpedance(frequency=LC_designer.frequency, name='ind',
z0=50,
```

```
                                Z=1j*LC_designer.frequency.w*24e-9)

# NB: it is also possible to create 2-port lumped elements like:
# line = rf.media.DefinedGammaZ0(frequency=LC_designer.frequency, z0=50)
# cap = line.capacitor(70e-12, name='cap')
# ind = line.inductor(24e-9, name='ind')

connections = [
    [(port1, 0), (cap, 0)],
    [(cap, 1), (ind, 0)],
    [(ind, 1), (port2, 0)]
]
circuit = rf.Circuit(connections)
LC_from_circuit = circuit.network

LC_from_circuit.plot_s_db(m=0, n=0, lw=2)
LC_from_circuit.plot_s_db(m=1, n=0, lw=2)
LC_from_circuit.plot_s_db(m=0, n=0, lw=2, ls='-.')
LC_from_circuit.plot_s_db(m=1, n=0, lw=2, ls='-.')
plt.show()
```

Smith Chart Example

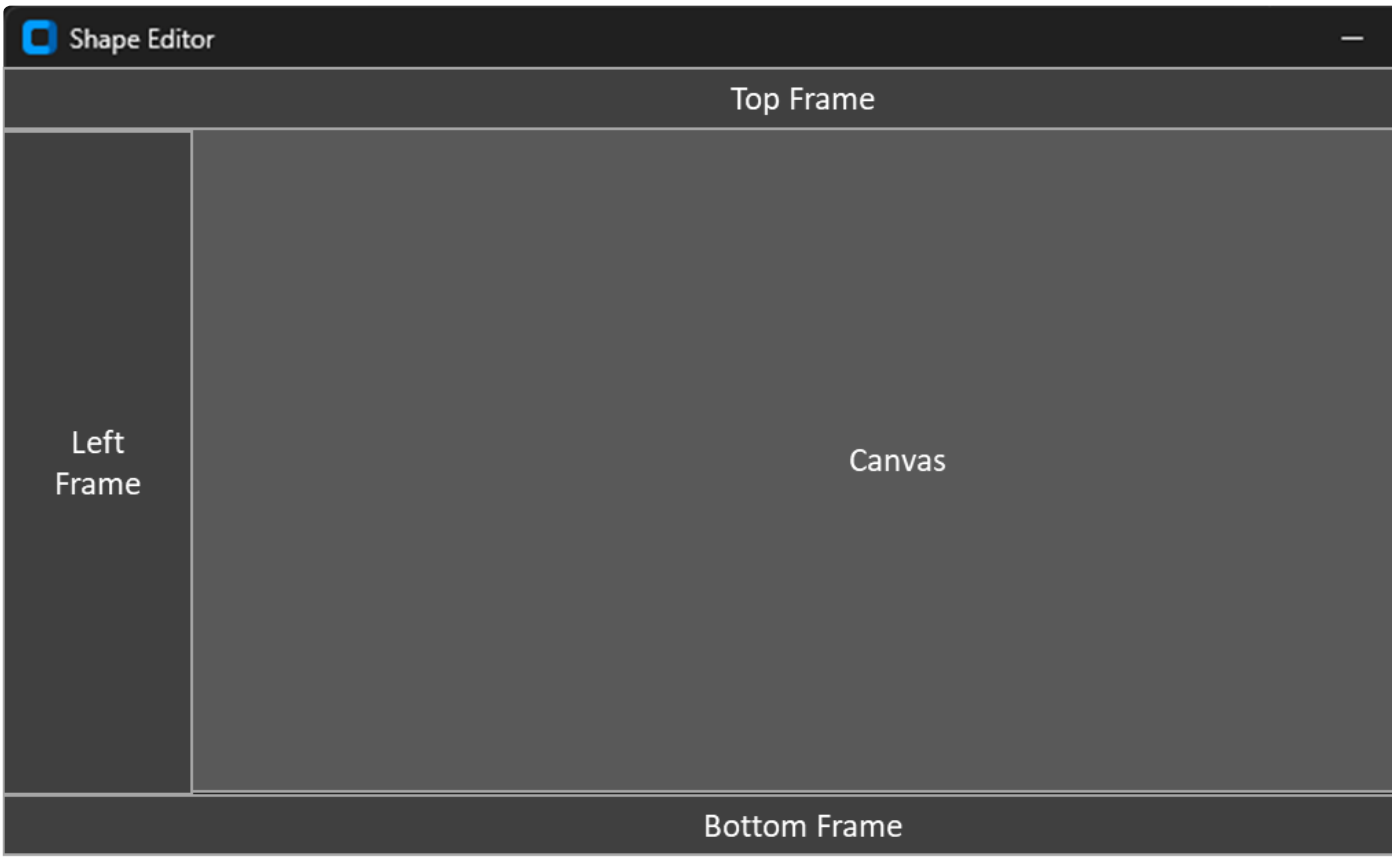


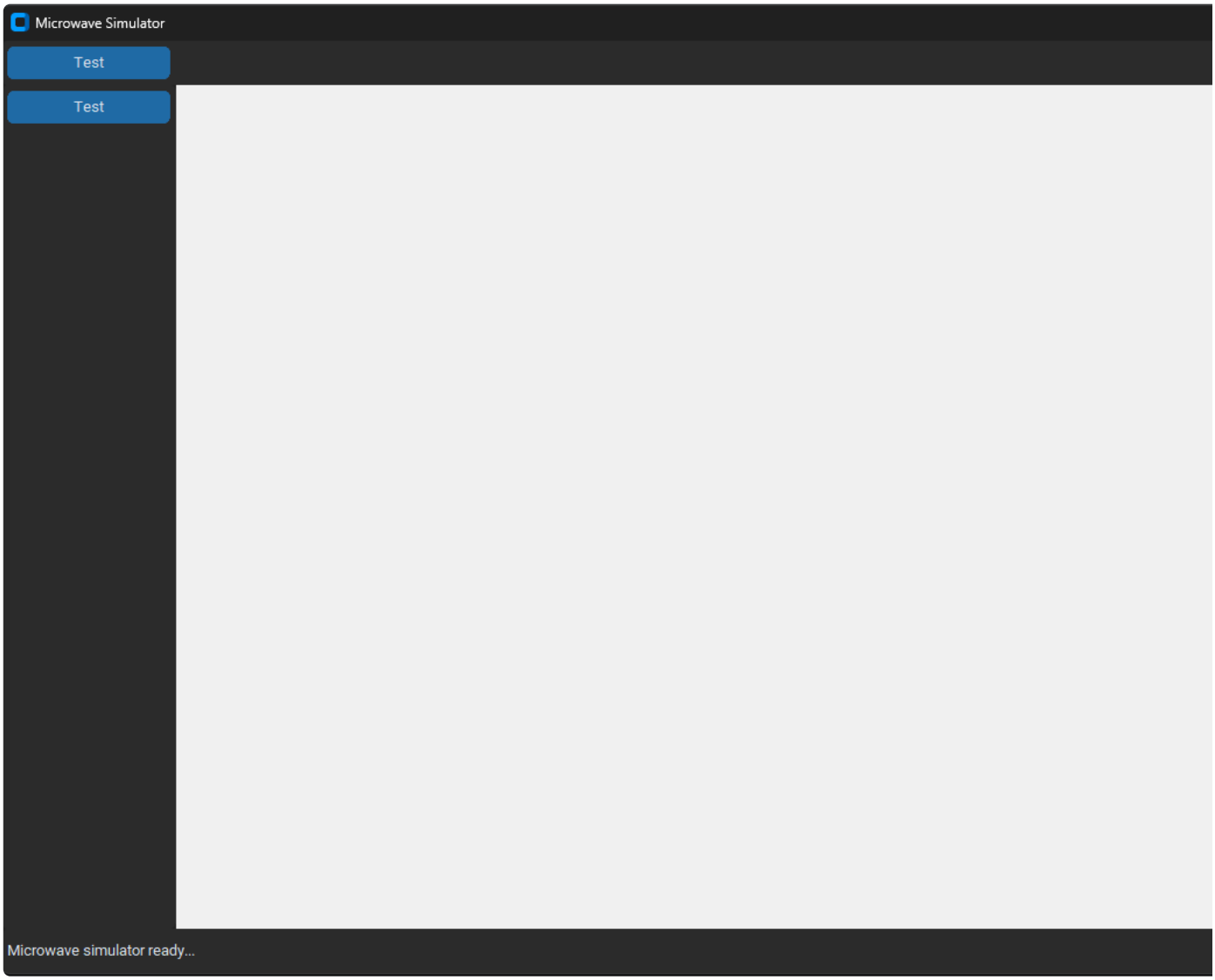
Lumped/smith_chart_example.py

```
import skrf as rf
import numpy as np
import matplotlib.pyplot as plt

network = rf.Network('data/ring slot.s2p')
network.plot_s_smith()

frequencies = np.linspace(0, 200e9, 201)
network.plot_s_smith()
plt.show()
```

Main User Interface

The initial user interface design should look familiar by now. It is the same UI layout used for the digital simulator.


Shape Editor
Top Frame
Left Frame
Canvas
Bottom Frame



microwave_simulator.py

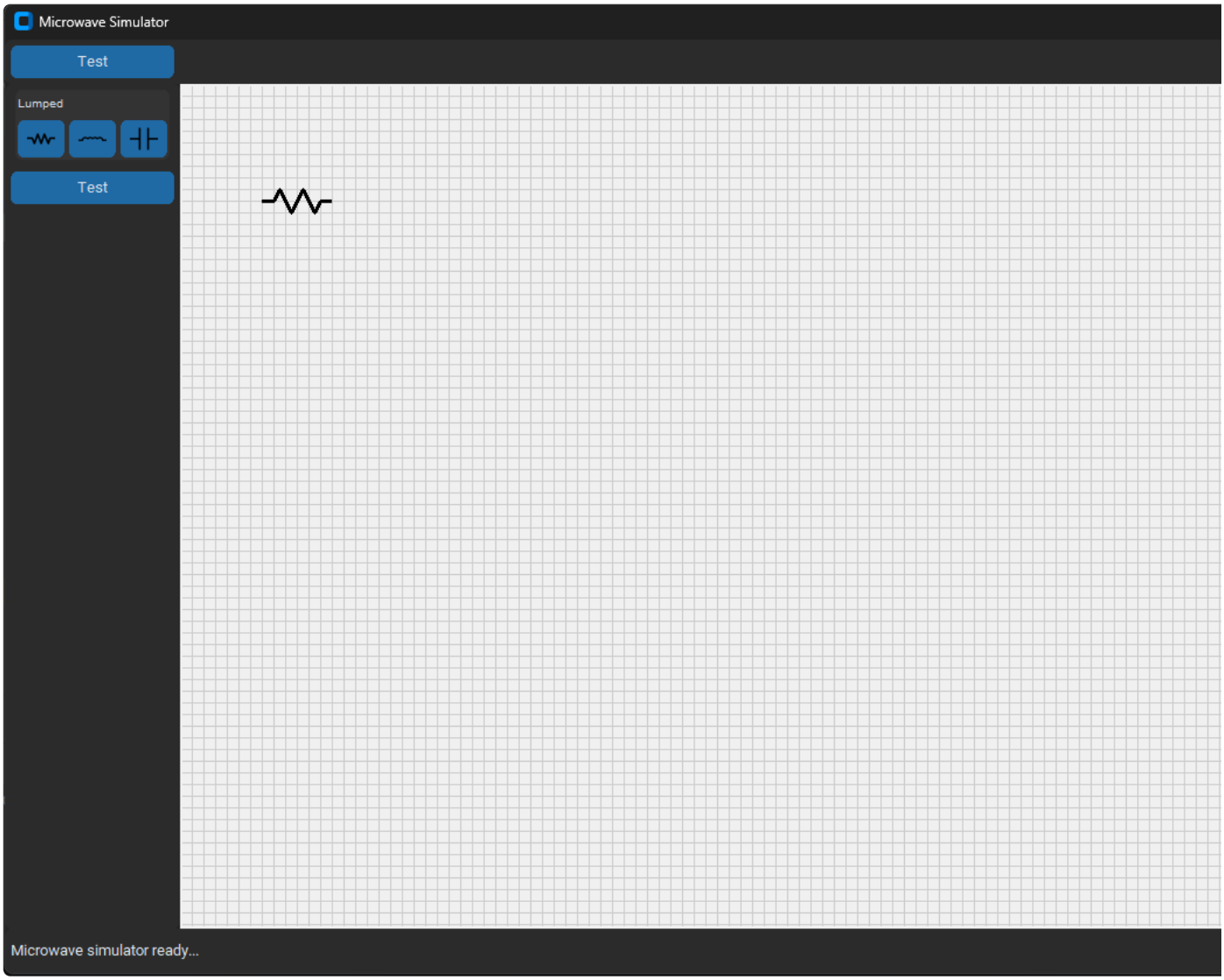
```
import customtkinter as ctk

ctk.set_appearance_mode("System")  # Modes: "System" (standard), "Dark",
"Light"
ctk.set_default_color_theme("blue")  # Themes: "blue" (standard), "green",
"dark-blue"


class MicrowaveSimulatorApp(ctk.CTk):
    """ctk.CTk is a CustomTkinter main window, similar to tk.Tk tkinter main
window"""

    def __init__(self):
        super().__init__()
        self.geometry("1200x800x100x100")  # w, h, x, y
```

```
        self.title("Microwave Simulator")

        self.canvas = ctk.CTkCanvas(self)
        self.top_frame = ctk.CTkFrame(self)
        self.left_frame = ctk.CTkFrame(self)
        self.bottom_frame = ctk.CTkFrame(self)

        self.top_frame.pack(side=ctk.TOP, fill=ctk.BOTH)
        self.bottom_frame.pack(side=ctk.BOTTOM, fill=ctk.BOTH)
        self.left_frame.pack(side=ctk.LEFT, fill=ctk.BOTH)
        self.canvas.pack(side=ctk.LEFT, fill=ctk.BOTH, expand=True)

        # Add widgets to frames to cause the frames to auto-size
        top_frame_button = ctk.CTkButton(self.top_frame, text="Test")
        top_frame_button.pack(side=ctk.LEFT, padx=5, pady=5)

        left_frame_button = ctk.CTkButton(self.left_frame, text="Test")
        left_frame_button.pack(side=ctk.TOP, padx=5, pady=5)

        bottom_frame_label = ctk.CTkLabel(self.bottom_frame, text="Microwave
simulator ready...")
        bottom_frame_label.pack(side=ctk.LEFT, padx=5, pady=5)


if __name__ == "__main__":
    """Instantiate the Digital Simulator app and run the main loop"""
    app = MicrowaveSimulatorApp()
    app.mainloop()
```

Lumped Element Classes

Objectives:

- Create a lumped Resistor Class
- Create a lumped Inductor Class
- Create a lumped Capacitor Class



Comp_Lib/resistor.py

```
class Resistor:
    """Model for lumped resistor"""
    def __init__(self, canvas, x1, y1):
        self.canvas = canvas
        self.x1, self.y1 = x1, y1
        self.w = 60
        self.h = 40

        self.type = "resistor"
        self.id = None
        self.bbox = None
        self.points = []

        self.create_resistor()
        self.update_bbox()
```

```
    def create_resistor(self):
        # Assume comp center is at x1, y1
        self.points = [
            self.x1 - self.w/2, self.y1, self.x1 - self.w/2 + 10, self.y1,  # input lead
            self.x1 - self.w / 2 + 15, self.y1 - 10, self.x1 - self.w/2 + 25, self.y1 + 10,  # Bump 1
            self.x1 - self.w / 2 + 35, self.y1 - 10, self.x1 - self.w / 2 + 45, self.y1 + 10,  # Bump 2
            self.x1 + self.w/2 - 10, self.y1, self.x1 + self.w/2, self.y1  # output lead
        ]
        self.id = self.canvas.create_line(self.points, fill="black", width=3, tags='comp')

    def update(self):
        pass

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = self.canvas.bbox(self.id)
```

Comp_Lib/inductor.py

```
import tkinter as tk


class Inductor:
    """Model for lumped inductor"""
    def __init__(self, canvas, x1, y1):
        self.canvas = canvas
        self.x1, self.y1 = x1, y1
        self.w = 60
        self.h = 40

        self.type = "inductor"
        self.id = None
        self.bbox = None
        self.ids = []
        self.points = []

        self.create_inductor()
```

```
        self.update_bbox()

    def create_inductor(self):
        # Assume comp center is at x1, y1
        self.points = [
            self.x1 - self.w/2, self.y1, self.x1 - self.w/2 + 10, self.y1,  # 
input lead
            self.x1 - self.w/2 + 10, self.y1 - 10
        ]
        self.id = self.canvas.create_line(self.points, fill="black", width=3, 
tags='comp')
        self.ids.append(self.id)

        self.points = [
            self.x1 + self.w / 2 - 10, self.y1 - 10,
            self.x1 + self.w / 2 - 10, self.y1, self.x1 + self.w / 2, self.y1
# output lead
        ]
        self.id = self.canvas.create_line(self.points, fill="black", width=3, 
tags='comp')
        self.ids.append(self.id)

        self.create_lines()
        self.create_arcs()

    def create_lines(self):
        for x in range(2, 5):
            self.points = [
                self.x1 - self.w / 2 + 10 * x, self.y1 - 10,
                self.x1 - self.w / 2 + 10 * x, self.y1
            ]
            self.id = self.canvas.create_line(self.points, fill="black", 
width=3, tags='comp')
            self.ids.append(self.id)

    def create_arcs(self):
        for x in range(2, 6):
            self.points = [
                self.x1 - self.w / 2 + 10 * x, self.y1 - 12,
                self.x1 - self.w / 2 + 10 * (x-1), self.y1 - 2
            ]
            self.id = self.canvas.create_arc(self.points, start=0, extent=180, 
                                             fill="black", width=3, 
style=tk.ARC, tags='comp')
            self.ids.append(self.id)
```

```
    def update(self):
        pass

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = self.canvas.bbox(self.id)
```

Comp_Lib/capacitor.py

```
class Capacitor:
    """Model for lumped capacitor"""
    def __init__(self, canvas, x1, y1):
        self.canvas = canvas
        self.x1, self.y1 = x1, y1
        self.w = 60
        self.h = 40

        self.type = "capacitor"
        self.id = None
        self.bbox = None
        self.points = []
        self.ids = []

        self.create_capacitor()
        self.update_bbox()

    def create_capacitor(self):
        # Assume comp center is at x1, y1
        self.points = [
            self.x1 - self.w/2, self.y1, self.x1 - self.w/2 + 25, self.y1,  #
input lead
            self.x1 - self.w / 2 + 25, self.y1 - 15, self.x1 - self.w/2 + 25,
self.y1 + 15,
        ]
        self.id = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')
        self.ids.append(self.id)

        self.points = [
            self.x1 + self.w/2, self.y1, self.x1 + self.w/2 - 25, self.y1,  #
output lead
            self.x1 + self.w / 2 - 25, self.y1 - 15, self.x1 + self.w/2 - 25,
self.y1 + 15,
```

```
        ]
        self.id = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')
        self.ids.append(self.id)

    def update(self):
        pass

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = self.canvas.bbox(self.id)
```

UI_Lib/lump_button_frame.py

```
import customtkinter as ctk
from tktooltip import ToolTip
from pathlib import Path
from PIL import Image

from Comp_Lib import Resistor, Inductor, Capacitor


class LumpButtonFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.button_id = None

        self.button_list = [("resistor", "../icons/resistor.png"),
                            ("inductor", "../icons/inductor.png"),
                            ("capacitor", "../icons/capacitor.png")]

        self.init_frame_widgets()

    def init_frame_widgets(self):
        frame_name_label = ctk.CTkLabel(self, text="Lumped", font=("Helvetica",
10), height=20)
        frame_name_label.grid(row=0, column=0, columnspan=3, sticky=ctk.W,
padx=2, pady=2)

        row_num, col_num = 1, 0
        for button in self.button_list:
```

```
            a_image = ctk.CTkImage(light_image=Image.open
                                   (Path(__file__).parent / button[1]),
                                   dark_image=Image.open
                                   (Path(__file__).parent / button[1]),
                                   size=(24, 24))
            self.button_id = ctk.CTkButton(self, text="", image=a_image,
width=30,
                                           command=lambda
a_name=button[0]:self.create_events(a_name))
            self.button_id.grid(row=row_num, column=col_num, sticky=ctk.W,
padx=2, pady=2)
            ToolTip(self.button_id, msg=button[0])
            row_num, col_num = self.update_grid_numbers(row_num, col_num)

    def create_events(self, name):
        gate = None
        if name == "resistor":
            gate = Resistor(self.canvas, 100, 100)
        elif name == "inductor":
            gate = Inductor(self.canvas, 100, 100)
        elif name == "capacitor":
            gate = Capacitor(self.canvas, 100, 100)
        self.canvas.comp_list.append(gate)
        self.canvas.redraw()
        self.canvas.mouse.move_mouse_bind_events()

    @staticmethod
    def update_grid_numbers(row, column):
        column += 1
        if column > 2:
            column = 0
            row += 1
        return row, column
```

UI_Lib/left_frame.py

```
import customtkinter as ctk
from UI_Lib import LumpButtonFrame


class LeftFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
```

```
        self.canvas = canvas

        self.comp_button_frame = LumpButtonFrame(self, self.canvas)
        self.comp_button_frame.pack(side=ctk.TOP, padx=5, pady=5)
```

microwave_simulator.py

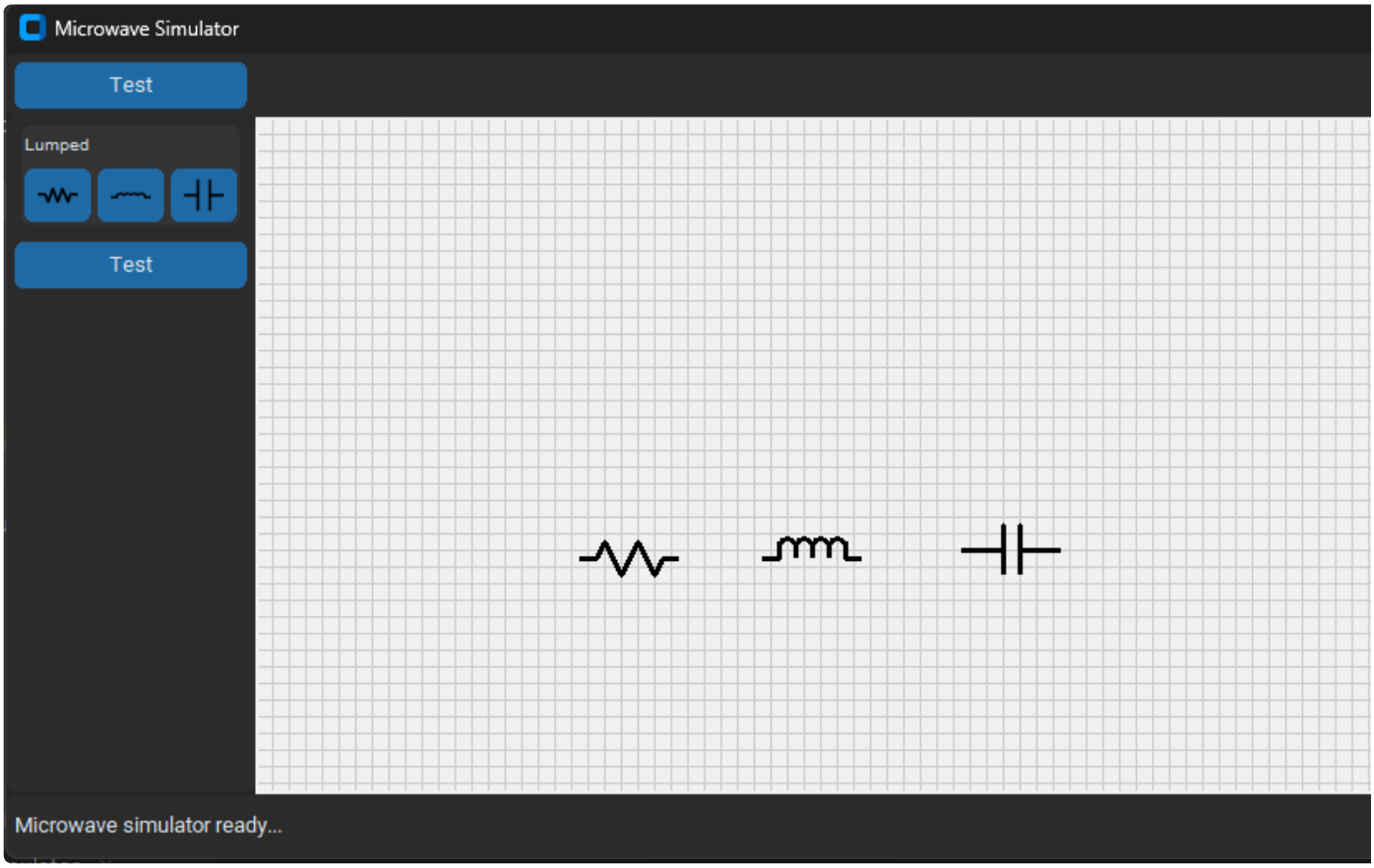
```
import customtkinter as ctk

from UI_Lib import LeftFrame, Canvas

ctk.set_appearance_mode("System")  # Modes: "System" (standard), "Dark",
"Light"
ctk.set_default_color_theme("blue")  # Themes: "blue" (standard), "green",
"dark-blue"


class MicrowaveSimulatorApp(ctk.CTk):
    """ctk.CTk is a CustomTkinter main window, similar to tk.Tk tkinter main
window"""

    def __init__(self):
        super().__init__()
        self.geometry("1200x800x100x100")  # w, h, x, y
        self.title("Microwave Simulator")

        self.canvas = Canvas(self)
        self.top_frame = ctk.CTkFrame(self)
        self.left_frame = LeftFrame(self, self.canvas)
        self.bottom_frame = ctk.CTkFrame(self)

        self.top_frame.pack(side=ctk.TOP, fill=ctk.BOTH)
        self.bottom_frame.pack(side=ctk.BOTTOM, fill=ctk.BOTH)
        self.left_frame.pack(side=ctk.LEFT, fill=ctk.BOTH)
        self.canvas.pack(side=ctk.LEFT, fill=ctk.BOTH, expand=True)

        # Add widgets to frames to cause the frames to auto-size
        top_frame_button = ctk.CTkButton(self.top_frame, text="Test")
        top_frame_button.pack(side=ctk.LEFT, padx=5, pady=5)

        left_frame_button = ctk.CTkButton(self.left_frame, text="Test")
        left_frame_button.pack(side=ctk.TOP, padx=5, pady=5)
```

```
        bottom_frame_label = ctk.CTkLabel(self.bottom_frame, text="Microwave
simulator ready...")
        bottom_frame_label.pack(side=ctk.LEFT, padx=5, pady=5)


if __name__ == "__main__":
    """Instantiate the Digital Simulator app and run the main loop"""
    app = MicrowaveSimulatorApp()
    app.mainloop()
```

Move Lumped Elements with Mouse

Objectives:

- Add selectors to lumped element classes
- Create a base class for lumped elements
- Add update methods to lumped element classes



Comp_Lib/component.py

```
import tkinter as tk
from PIL import Image, ImageTk


class Comp:
    def __init__(self, canvas, x1, y1):
        """Base class for gate classes"""
        self.canvas = canvas
        self.x1 = x1
        self.y1 = y1

        self.id = None
        self.sel_id = None
        self.a_image = None
        self.ph_image = None
        self.bbox = None
        self.angle = 0
        self.filename = None

        self.is_selected = False
        self.is_drawing = False

        self.in1, self.in2, self.out = None, None, None
        self.conn_list = []
        self.wire_list = []

    def update_position(self):
        """Update the position when the gate object is moved"""
        self.canvas.coords(self.id, self.x1, self.y1)  # Update position

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = self.canvas.bbox(self.id)

    def create_selector(self):
        """Create the red rectangle selector and check to see if the gate is
selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5,
self.bbox[3] + 5
        self.sel_id = self.canvas.create_rectangle(x1, y1, x2, y2, fill=None,
outline="red", width=2)
        self.set_selector_visibility()

    def update_selector(self):
        """Update the red rectangle selector coordinates and check to see if
the gate is selected"""
```

```
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5,
self.bbox[3] + 5
        self.canvas.coords(self.sel_id, x1, y1, x2, y2)
        self.set_selector_visibility()

    def set_selector_visibility(self):
        """Set the selector visibility state"""
        if self.is_selected:
            self.canvas.itemconfig(self.sel_id, state='normal')
        else:
            self.canvas.itemconfig(self.sel_id, state='hidden')

    def set_connector_visibility(self):
        """Set the connector visibility state"""
        if self.is_drawing:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='normal')
        else:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='hidden')

    def rotate(self):
        """Set the rotation angle to the current angle + 90 deg, reset to 0 deg
if angle > 270 deg"""
        self.angle += 90
        if self.angle > 270:
            self.angle = 0

    def check_connector_hit(self, x, y):
        """Hit test to see if a connector is at the provided x, y
coordinates"""
        for conn in self.conn_list:
            if conn.connector_hit_test(x, y):
                return conn
        return None

    def move_connected_wires(self):
        """Resize connected wires if the shape is moved"""
        for connection in self.wire_list:
            for connector in self.conn_list:
                if connector == connection.connector_obj:
                    # print(connector, connection.line_obj, "Match")
                    if connection.wire_end == "begin":
                        connection.wire_obj.x1 = connector.x
                        connection.wire_obj.y1 = connector.y
                    elif connection.wire_end == "end":
```

```
                        connection.wire_obj.x2 = connector.x
                        connection.wire_obj.y2 = connector.y
```

Comp_Lib/resistor.py

```
from Comp_Lib.component import Comp


class Resistor(Comp):
    """Model for lumped resistor"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.w = 60
        self.h = 40

        self.type = "resistor"
        self.points = []

        self.create_resistor()
        self.update_bbox()
        self.create_selector()

    def create_resistor(self):
        # Assume comp center is at x1, y1
        self.points = [
            self.x1 - self.w/2, self.y1, self.x1 - self.w/2 + 10, self.y1,  #
input lead
            self.x1 - self.w / 2 + 15, self.y1 - 10, self.x1 - self.w/2 + 25,
self.y1 + 10,  # Bump 1
            self.x1 - self.w / 2 + 35, self.y1 - 10, self.x1 - self.w / 2 + 45,
self.y1 + 10,  # Bump 2
            self.x1 + self.w/2 - 10, self.y1, self.x1 + self.w/2, self.y1  #
output lead
        ]
        self.id = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')

    def update(self):
        self.update_position()
        self.update_bbox()
        self.update_selector()

    def update_position(self):
```

```
        self.points = [
            self.x1 - self.w/2, self.y1, self.x1 - self.w/2 + 10, self.y1,  # input lead
            self.x1 - self.w / 2 + 15, self.y1 - 10, self.x1 - self.w/2 + 25, self.y1 + 10,  # Bump 1
            self.x1 - self.w / 2 + 35, self.y1 - 10, self.x1 - self.w / 2 + 45, self.y1 + 10,  # Bump 2
            self.x1 + self.w/2 - 10, self.y1, self.x1 + self.w/2, self.y1  # output lead
        ]
        self.canvas.coords(self.id, self.points)

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = [self.x1 - self.w/2, self.y1 - self.h/2, self.x1 + self.w/2, self.y1 + self.h/2]
```

Comp_Lib/inductor.py

```
import tkinter as tk
from Comp_Lib.component import Comp


class Inductor(Comp):
    """Model for lumped inductor"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.w = 60
        self.h = 40

        self.type = "inductor"
        self.points = []
        self.id1, self.id2, self.id3, self.id4, self.id5, self.id6, self.id7, self.id8, self.id9 = (
            None, None, None, None, None, None, None, None, None,)

        self.create_inductor()
        self.update_bbox()
        self.create_selector()

    def create_inductor(self):
        # Assume comp center is at x1, y1
        self.points = [
```

```
            self.x1 - self.w/2, self.y1, self.x1 - self.w/2 + 10, self.y1,  #
input lead
            self.x1 - self.w/2 + 10, self.y1 - 10
        ]
        self.id1 = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')

        self.points = [
            self.x1 + self.w / 2 - 10, self.y1 - 10,
            self.x1 + self.w / 2 - 10, self.y1, self.x1 + self.w / 2, self.y1
# output lead
        ]
        self.id2 = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')

        self.create_lines()
        self.create_arcs()

    def create_lines(self):
        self.points = [
            self.x1 - self.w / 2 + 10 * 2, self.y1 - 10,
            self.x1 - self.w / 2 + 10 * 2, self.y1
        ]
        self.id3 = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')

        self.points = [
            self.x1 - self.w / 2 + 10 * 3, self.y1 - 10,
            self.x1 - self.w / 2 + 10 * 3, self.y1
        ]
        self.id4 = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')

        self.points = [
            self.x1 - self.w / 2 + 10 * 4, self.y1 - 10,
            self.x1 - self.w / 2 + 10 * 4, self.y1
        ]
        self.id5 = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')

    def create_arcs(self):
        self.points = [
            self.x1 - self.w / 2 + 10 * 2, self.y1 - 12,
            self.x1 - self.w / 2 + 10 * (2-1), self.y1 - 2
        ]
        self.id6 = self.canvas.create_arc(self.points, start=0, extent=180,
```

```
                                        fill="black", width=3, style=tk.ARC,
tags='comp')

        self.points = [
            self.x1 - self.w / 2 + 10 * 3, self.y1 - 12,
            self.x1 - self.w / 2 + 10 * (3-1), self.y1 - 2
        ]
        self.id7 = self.canvas.create_arc(self.points, start=0, extent=180,
                                        fill="black", width=3, style=tk.ARC,
tags='comp')

        self.points = [
            self.x1 - self.w / 2 + 10 * 4, self.y1 - 12,
            self.x1 - self.w / 2 + 10 * (4-1), self.y1 - 2
        ]
        self.id8 = self.canvas.create_arc(self.points, start=0, extent=180,
                                        fill="black", width=3, style=tk.ARC,
tags='comp')

        self.points = [
            self.x1 - self.w / 2 + 10 * 5, self.y1 - 12,
            self.x1 - self.w / 2 + 10 * (5-1), self.y1 - 2
        ]
        self.id9 = self.canvas.create_arc(self.points, start=0, extent=180,
                                        fill="black", width=3, style=tk.ARC,
tags='comp')

    def update(self):
        self.update_position()
        self.update_bbox()
        self.update_selector()

    def update_position(self):
        # Update lead positions
        self.points = [
            self.x1 - self.w / 2, self.y1, self.x1 - self.w / 2 + 10, self.y1,
# input lead
            self.x1 - self.w / 2 + 10, self.y1 - 10
        ]
        self.canvas.coords(self.id1, self.points)

        self.points = [
            self.x1 + self.w / 2 - 10, self.y1 - 10,
            self.x1 + self.w / 2 - 10, self.y1, self.x1 + self.w / 2, self.y1
# output lead
        ]
```

```
        self.canvas.coords(self.id2, self.points)

        # Update line positions
        self.points = [
            self.x1 - self.w / 2 + 10 * 2, self.y1 - 10,
            self.x1 - self.w / 2 + 10 * 2, self.y1
        ]
        self.canvas.coords(self.id3, self.points)

        self.points = [
            self.x1 - self.w / 2 + 10 * 3, self.y1 - 10,
            self.x1 - self.w / 2 + 10 * 3, self.y1
        ]
        self.canvas.coords(self.id4, self.points)

        self.points = [
            self.x1 - self.w / 2 + 10 * 4, self.y1 - 10,
            self.x1 - self.w / 2 + 10 * 4, self.y1
        ]
        self.canvas.coords(self.id5, self.points)

        # Update arc positions
        self.points = [
            self.x1 - self.w / 2 + 10 * 2, self.y1 - 12,
            self.x1 - self.w / 2 + 10 * (2-1), self.y1 - 2
        ]
        self.canvas.coords(self.id6, self.points)

        self.points = [
            self.x1 - self.w / 2 + 10 * 3, self.y1 - 12,
            self.x1 - self.w / 2 + 10 * (3-1), self.y1 - 2
        ]
        self.canvas.coords(self.id7, self.points)

        self.points = [
            self.x1 - self.w / 2 + 10 * 4, self.y1 - 12,
            self.x1 - self.w / 2 + 10 * (4-1), self.y1 - 2
        ]
        self.canvas.coords(self.id8, self.points)

        self.points = [
            self.x1 - self.w / 2 + 10 * 5, self.y1 - 12,
            self.x1 - self.w / 2 + 10 * (5-1), self.y1 - 2
        ]
        self.canvas.coords(self.id9, self.points)
```

```
    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = [self.x1 - self.w/2, self.y1 - self.h/2, self.x1 +
self.w/2, self.y1 + self.h/2]
```

Comp_Lib/capacitor.py

```
from Comp_Lib.component import Comp


class Capacitor(Comp):
    """Model for lumped capacitor"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.w = 60
        self.h = 40

        self.type = "capacitor"
        self.points = []

        self.id1, self.id2 = None, None

        self.create_capacitor()
        self.update_bbox()
        self.create_selector()

    def create_capacitor(self):
        # Assume comp center is at x1, y1
        self.points = [
            self.x1 - self.w/2, self.y1, self.x1 - self.w/2 + 25, self.y1,  #
input lead
            self.x1 - self.w / 2 + 25, self.y1 - 15, self.x1 - self.w/2 + 25,
self.y1 + 15,
        ]
        self.id1 = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')

        self.points = [
            self.x1 + self.w/2, self.y1, self.x1 + self.w/2 - 25, self.y1,  #
output lead
            self.x1 + self.w / 2 - 25, self.y1 - 15, self.x1 + self.w/2 - 25,
self.y1 + 15,
        ]
```

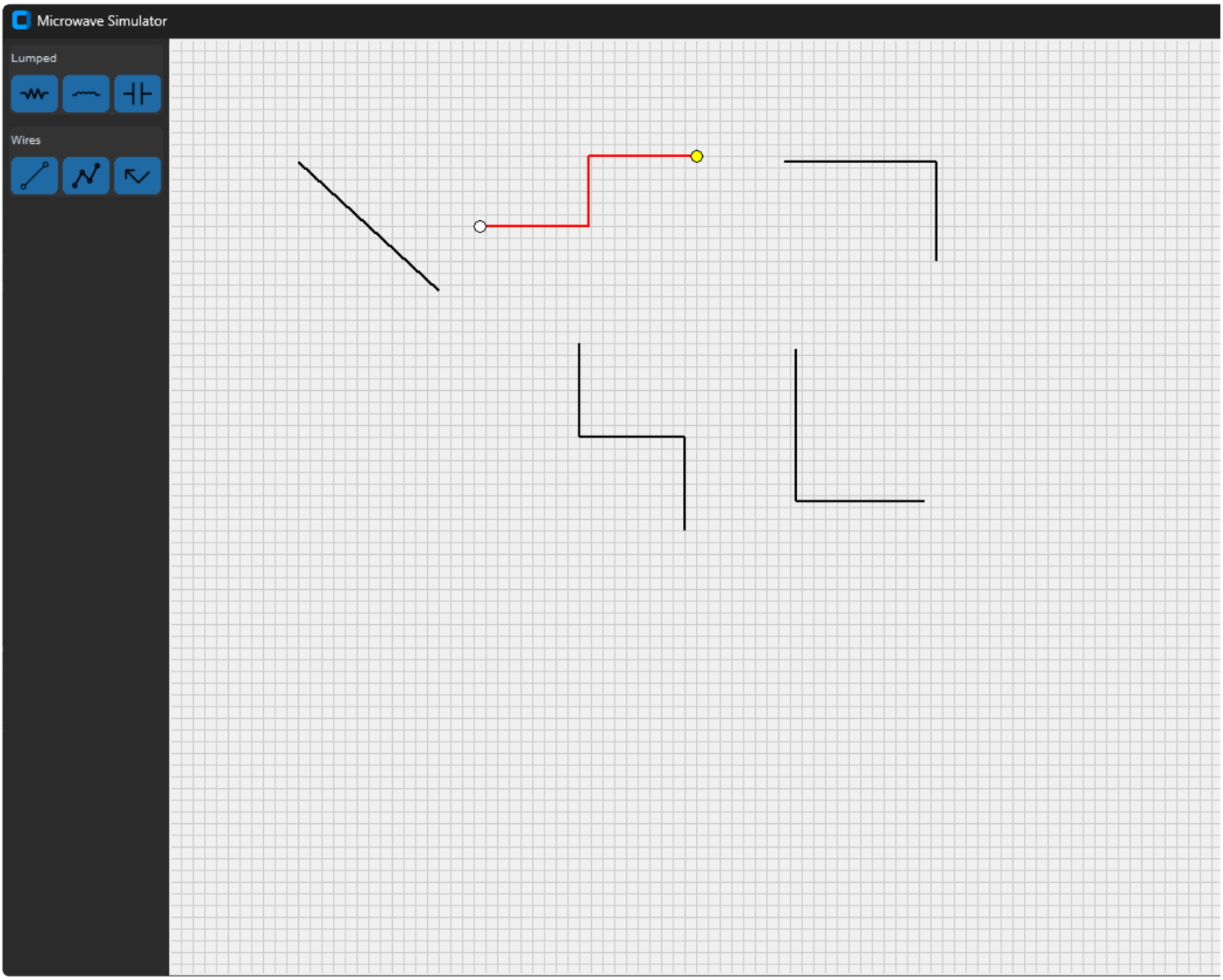
```
        self.id2 = self.canvas.create_line(self.points, fill="black", width=3,
tags='comp')

    def update(self):
        self.update_position()
        self.update_bbox()
        self.update_selector()

    def update_position(self):
        self.points = [
            self.x1 - self.w/2, self.y1, self.x1 - self.w/2 + 25, self.y1,  #
input lead
            self.x1 - self.w / 2 + 25, self.y1 - 15, self.x1 - self.w/2 + 25,
self.y1 + 15,
        ]
        self.canvas.coords(self.id1, self.points)

        self.points = [
            self.x1 + self.w/2, self.y1, self.x1 + self.w/2 - 25, self.y1,  #
output lead
            self.x1 + self.w / 2 - 25, self.y1 - 15, self.x1 + self.w/2 - 25,
self.y1 + 15,
        ]
        self.canvas.coords(self.id2, self.points)

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = [self.x1 - self.w/2, self.y1 - self.h/2, self.x1 +
self.w/2, self.y1 + self.h/2]
```

Wire Classes

Objectives:

- Straight wire class
  - Straight wire hit test based on equation of a line between two points
- 3-Segment Wire Class
  - Horizontal and vertical orientation based on line direction when drawing
- 2-Segment Elbow Class
  - Horizontal and vertical orientation based on line direction when drawing
- Wire select and move with mouse

- End selector color change when selected
- Wire resize with mouse



`Wire_Lib` / `__init__.py`

```
# import wire related classes
from Wire_Lib.straight_wire import StraightWire
from Wire_Lib.segment_wire import SegmentWire
from Wire_Lib.elbow_wire import ElbowWire

# import wire selector related classes
from Wire_Lib.wire_selector import WireSelector
```

`Wire_Lib` /wire.py - base class for wire classes

```python
from Wire_Lib.wire_selector import WireSelector


class Wire:
    """Base class for wire classes"""
    def __init__(self, canvas, x1, y1, x2, y2):
        self.canvas = canvas
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

        self.fill_color = "black"
        self.border_width = 2
        self.line_direction = "horizontal"

        self.id = None
        self.is_selected = False
        self.selector = None
        self.width = 2
        self.bbox = None

        self.sel1, self.sel2 = None, None

    def update_bbox(self):
        self.bbox = self.canvas.bbox(self.id)

    def create_selectors(self):
        self.sel1 = WireSelector(self.canvas, "begin", self.x1, self.y1)
        self.sel2 = WireSelector(self.canvas, "end", self.x2, self.y2)

    def update_selectors(self):
        self.sel1.x, self.sel1.y = self.x1, self.y1
        self.sel2.x, self.sel2.y = self.x2, self.y2
        self.sel1.update()
        self.sel2.update()

    def update_selection(self):
        if self.is_selected:
            self.canvas.itemconfigure(self.id, fill="red")
            self.canvas.itemconfigure(self.sel1.id, state='normal')
            self.canvas.itemconfigure(self.sel2.id, state='normal')
        else:
            self.canvas.itemconfigure(self.id, fill="black")
            self.canvas.itemconfigure(self.sel1.id, state='hidden')
            self.canvas.itemconfigure(self.sel2.id, state='hidden')
```

```
    def hit_test(self, x, y):
        x1, y1 = self.bbox[0], self.bbox[1]
        x2, y2 = self.bbox[2], self.bbox[3]
        if x1 <= x <= x2 and y1 <= y <= y2:
            self.is_selected = True
        else:
            self.is_selected = False

    def sel_hit_test(self, x, y):
        if self.sel1.selector_hit_test(x, y):
            self.selector = self.sel1.name
            return self.sel1
        elif self.sel2.selector_hit_test(x, y):
            self.selector = self.sel2.name
            return self.sel2
        else:
            return None

    def resize(self, offsets, event):
        offset_x1, offset_y1, offset_x2, offset_y2 = offsets
        if self.selector == "end":
            x2 = event.x - offset_x2
            y2 = event.y - offset_y2
            self.x2, self.y2 = x2, y2
            self.x2, self.y2 = self.canvas.grid.snap_to_grid(self.x2, self.y2)
        elif self.selector == "begin":
            x1 = event.x - offset_x1
            y1 = event.y - offset_y1
            self.x1, self.y1 = x1, y1
            self.x1, self.y1 = self.canvas.grid.snap_to_grid(self.x1, self.y1)
```

Wire_Lib/straight_wire.py

```
from Wire_Lib.wire import Wire


class StraightWire(Wire):
    def __init__(self, canvas, x1, y1, x2, y2):
        super().__init__(canvas, x1, y1, x2, y2)

        self.create_wire()
        self.create_selectors()
```

```
        self.update_selection()

    def create_wire(self):
        self.id = self.canvas.create_line(self.x1, self.y1, self.x2, self.y2,
width=self.width)

    def update(self):
        self.update_position()
        self.update_selectors()
        self.update_selection()

    def update_position(self):
        self.canvas.coords(self.id, self.x1, self.y1, self.x2, self.y2)

    def hit_test(self, x, y):
        # 2-Point Line equation: y = m * (x - x1) + y1
        x1, y1 = self.x1, self.y1
        x2, y2 = self.x2, self.y2

        # Calculate the slope: m = (y2 - y1) / (x2 - x1)
        if (x2 - x1) == 0:
            m = 0
        else:
            m = (y2 - y1)/(x2 - x1)

        # Check to see if the point (x, y) is on the line and between the two
end points
        tol = 10
        if y - tol <= m*(x - x1) + y1 <= y + tol:
            if (min(x1, x2) <= x <= max(x1, x2)) and (min(y1, y2) <= y <=
max(y1, y2)):
                self.is_selected = True
        else:
            self.is_selected = False
```

Wire_Lib/segment_wire.py

```
from Wire_Lib.wire import Wire


class SegmentWire(Wire):
    def __init__(self, canvas, x1, y1, x2, y2):
        super().__init__(canvas, x1, y1, x2, y2)
```

```
        self.seg1, self.seg2, self.seg3 = None, None, None
        self.segment_list = []

        self.create_wire()
        self.update_bbox()
        self.create_selectors()
        self.update_selection()

    def create_wire(self):
        w = self.x2 - self.x1
        h = self.y2 - self.y1

        if abs(w) >= abs(h):  # Horizontal
            self.seg1 = self.x1, self.y1, self.x1 + w / 2, self.y1
            self.seg2 = self.x1 + w / 2, self.y1, self.x1 + w / 2, self.y2
            self.seg3 = self.x1 + w / 2, self.y2, self.x2, self.y2
        else:  # Vertical
            self.seg1 = self.x1, self.y1, self.x1, self.y1 + h / 2
            self.seg2 = self.x1, self.y1 + h / 2, self.x2, self.y1 + h / 2
            self.seg3 = self.x2, self.y1 + h / 2, self.x2, self.y2
        self.segment_list = [self.seg1, self.seg2, self.seg3]
        self.draw_segments()

    def draw_segments(self):
        for s in self.segment_list:
            self.id = self.canvas.create_line(s, fill=self.fill_color,
width=self.border_width, tags='wire')

    def update(self):
        self.update_position()
        self.update_bbox()
        self.update_selectors()
        self.update_selection()

    def update_position(self):
        """Update the position when the gate object is moved"""
        w = self.x2 - self.x1
        h = self.y2 - self.y1
        if abs(w) >= abs(h):
            self.canvas.coords(self.id, self.x1, self.y1, self.x1 + w / 2,
self.y1,
                               self.x1 + w / 2, self.y1, self.x1 + w / 2,
self.y2,
                               self.x1 + w / 2, self.y2, self.x2, self.y2)
        else:
```

```
            self.canvas.coords(self.id, self.x1, self.y1, self.x1, self.y1 + h
/ 2,
                               self.x1, self.y1 + h / 2, self.x2, self.y1 + h /
2,
                               self.x2, self.y1 + h / 2, self.x2, self.y2)
```

Wire_Lib/elbow_wire.py

```
from Wire_Lib.wire import Wire


class ElbowWire(Wire):
    def __init__(self, canvas, x1, y1, x2, y2):
        super().__init__(canvas, x1, y1, x2, y2)

        self.create_wire()
        self.update_bbox()
        self.create_selectors()
        self.update_selection()

    def create_wire(self):
        w = self.x2 - self.x1
        h = self.y2 - self.y1

        if abs(w) >= abs(h):  # Horizontal
            self.id = self.canvas.create_line(self.x1, self.y1, self.x2,
self.y1,
                                              self.x2, self.y1, self.x2,
self.y2,
                                              fill=self.fill_color,
                                              width=self.border_width,
tags="wire")
        else:  # Vertical
            self.id = self.canvas.create_line(self.x1, self.y1, self.x1,
self.y2,
                                              self.x1, self.y2, self.x2,
self.y2,
                                              fill=self.fill_color,
                                              width=self.border_width,
tags="wire")

    def update(self):
        self.update_position()
```

```
        self.update_bbox()
        self.update_selectors()
        self.update_selection()

    def update_position(self):
        """Update the position when the gate object is moved"""
        w = self.x2 - self.x1
        h = self.y2 - self.y1
        if abs(w) >= abs(h):
            self.canvas.coords(self.id, self.x1, self.y1, self.x2, self.y1,
                                        self.x2, self.y1, self.x2, self.y2)
        else:
            self.canvas.coords(self.id, self.x1, self.y1, self.x1, self.y2,
                                        self.x1, self.y2, self.x2, self.y2)
```

Wire_Lib/wire_selector.py

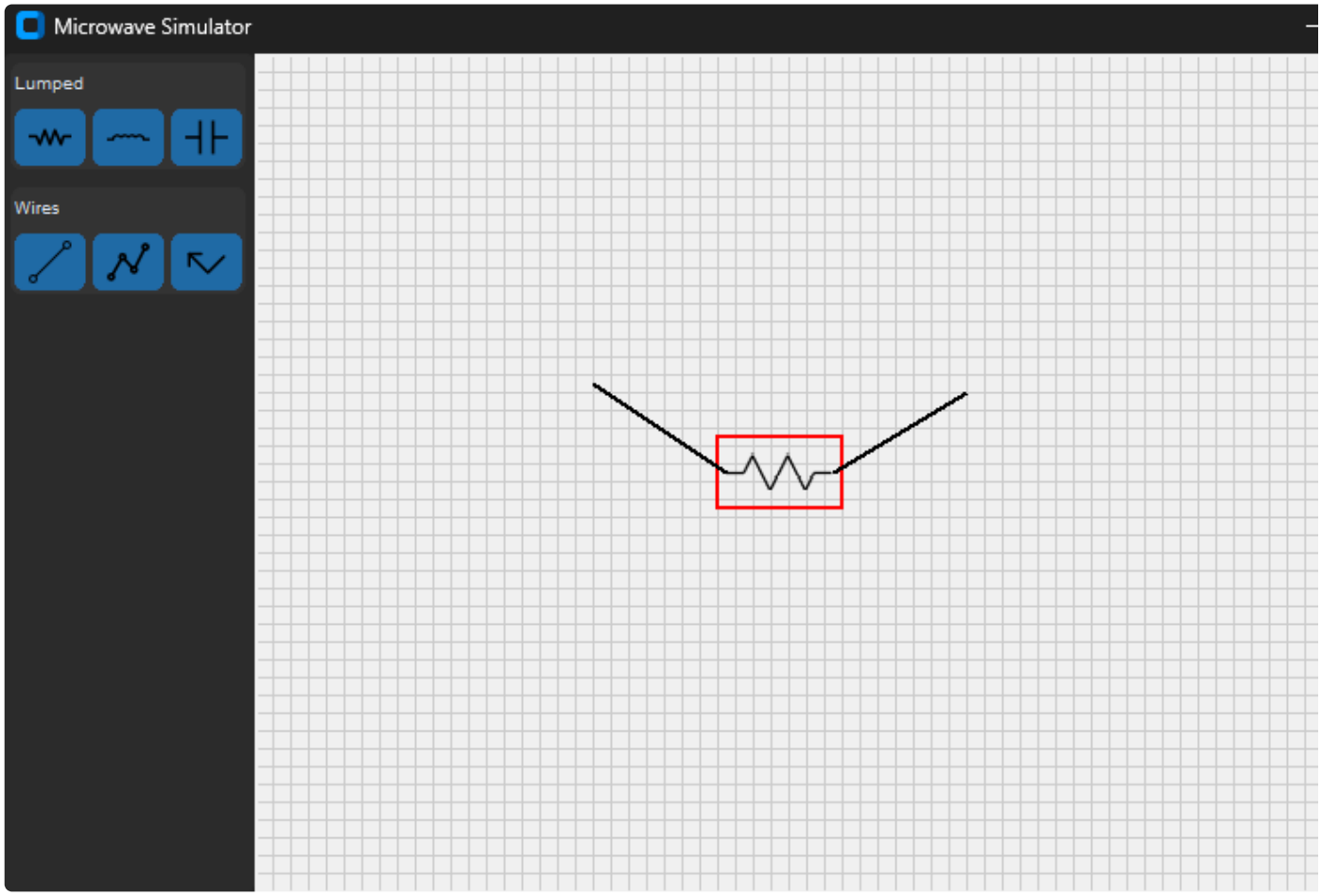
```
class WireSelector:
    def __init__(self, canvas, name, x, y):
        self.canvas = canvas
        self.name = name
        self.x = x
        self.y = y
        self.is_selected = False

        self.id = self.canvas.create_oval(self.x - 5, self.y - 5, self.x + 5,
self.y + 5,
                                          state='normal', fill="white")

    def update(self):
        self.update_position()
        self.update_selection()

    def update_position(self):
        """Update the selector position"""
        sel_points = [self.x - 5, self.y - 5, self.x + 5, self.y + 5]
        self.canvas.coords(self.id, sel_points)

    def update_selection(self):
        if self.is_selected:
            self.canvas.itemconfigure(self.id, fill="yellow")
        else:
            self.canvas.itemconfigure(self.id, fill="white")
```

```
    def selector_hit_test(self, event_x, event_y):
        if self.x-5 <= event_x <= self.x+5 and self.y-5 <= event_y <= self.y+5:
            self.is_selected = True
            self.update_selection()
            return True
        else:
            self.is_selected = False
            self.update_selection()
            return False
```

Connection Classes

Objectives:

- Connect wires to component "connectors"
- Add wire list to each component to keep track of connected wires
- Move connected wires when component moves with mouse



Comp_Lib/`__init__.py`

```
# Import lumped components
from Comp_Lib.resistor import Resistor
from Comp_Lib.inductor import Inductor
from Comp_Lib.capacitor import Capacitor

# Import connector related classes
from Comp_Lib.connector import Connector
from Comp_Lib.connection import Connection
```

Comp_Lib/connector.py

```
class Connector:
    def __init__(self, canvas, name, x, y):
        """Connector class"""
        self.canvas = canvas
        self.name = name
        self.x = x
        self.y = y

        self.id = None

        self.radius = 5
        self.x1, self.y1, self.x2, self.y2 = (self.x - self.radius, self.y -
self.radius,
                                              self.x + self.radius, self.y +
self.radius)

        self.create_connector()

    def create_connector(self):
        # Create the connector here
        points = [self.x - self.radius, self.y - self.radius, self.x +
self.radius, self.y + self.radius]
        self.id = self.canvas.create_oval(points, fill="white",
outline="black", width=2, tags='connector')

    def update(self):
        """Update the connector here"""
        self.x1, self.y1, self.x2, self.y2 = (self.x - self.radius, self.y -
self.radius,
```

```
                                            self.x + self.radius, self.y +
self.radius)
        points = [self.x - self.radius, self.y - self.radius, self.x +
self.radius, self.y + self.radius]
        self.canvas.coords(self.id, points)

    def set_pos(self, x, y):
        """Set the connector position here"""
        self.x = x
        self.y = y

    def connector_hit_test(self, x, y):
        """Connector hit test"""
        if self.x1 <= x <= self.x2 and self.y1 <= y <= self.y2:
            return True
        else:
            return False

    def __repr__(self):
        return ("Connector: " + self.name + " (" + str(self.x1) + ", " +
str(self.y1) + ")" +
                " (" + str(self.x2) + ", " + str(self.y2) + ")")
```

Comp_Lib/connection.py

```
class Connection:
    def __init__(self, conn_obj, wire_obj, wire_end):
        self.connector_obj = conn_obj
        self.wire_obj = wire_obj
        self.wire_end = wire_end      # "begin" or "end"

    def __repr__(self):
        return "Connection Object: " + str(self.connector_obj.name) + \
               " Connection Object Location: " + str(self.connector_obj.x) + ",
" + str(self.connector_obj.y) + \
               " Line Object: " + str(self.wire_obj) + \
               " Line End: " + self.wire_end
```

Comp_Lib/component.py

```
import tkinter as tk
from PIL import Image, ImageTk

from Helper_Lib import Point  # Import Point Class
from Comp_Lib.connector import Connector  # Import Connector Class


class Component:
    def __init__(self, canvas, x1, y1):
        """Base class for component classes"""
                self.canvas = canvas
        self.x1 = x1
        self.y1 = y1

        self.id = None
        self.sel_id = None
        self.is_selected = False
        self.is_drawing = False
        self.selector = None
        self.angle = 0

        self.a_image = None
        self.ph_image = None
        self.bbox = None

        self.conn_list = []  # Connector list
        self.wire_list = []  # Wire list (list of connections)
. . .

    def create_connectors(self):  # Connector method
        # Calculate position of connectors from current shape position and size
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w, h = x2 - x1, y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        # Define 3 connectors: in1, in2, out
        self.out = Connector(self.canvas, "out", center.x + w / 2, center.y)
        self.in1 = Connector(self.canvas, "in1", center.x - w / 2, center.y)

        # Update the connector list
        self.conn_list = [self.out, self.in1]

    def update_connectors(self):  # Connector method
        """Update the position of all connectors here"""
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w = x2 - x1
```

```
        h = y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        if self.angle == 0:
            self.out.x, self.out.y = center.x + w / 2, center.y
            self.in1.x, self.in1.y = center.x - w / 2, center.y
        elif self.angle == 90:
            self.out.x, self.out.y = center.x, center.y - h / 2
            self.in1.x, self.in1.y = center.x, center.y + h / 2
        elif self.angle == 180:
            self.out.x, self.out.y = center.x - w / 2, center.y
            self.in1.x, self.in1.y = center.x + w / 2, center.y
        elif self.angle == 270:
            self.out.x, self.out.y = center.x, center.y + h / 2
            self.in1.x, self.in1.y = center.x, center.y - h / 2

        for c in self.conn_list:
            c.update()

        self.move_connected_wires()

    def set_connector_visibility(self):  # Connector method
        """Set the connector visibility state"""
        if self.is_drawing:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='normal')
        else:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='hidden')

    def check_connector_hit(self, x, y):  # Connector method
        """Hit test to see if a connector is at the provided x, y
coordinates"""
        for conn in self.conn_list:
            if conn.connector_hit_test(x, y):
                return conn
        return None

    def move_connected_wires(self):  # Connector method
        """Resize connected wires if the shape is moved"""
        for connection in self.wire_list:
            for connector in self.conn_list:
                if connector == connection.connector_obj:
                    # print(connector, connection.line_obj, "Match")
                    if connection.wire_end == "begin":
                        connection.wire_obj.x1 = connector.x
```

```
                        connection.wire_obj.y1 = connector.y
                    elif connection.wire_end == "end":
                        connection.wire_obj.x2 = connector.x
                        connection.wire_obj.y2 = connector.y

. . .
```

Comp_Lib/resistor.py

```
from pathlib import Path

from Helper_Lib import Point
from Comp_Lib.component import Component
from Comp_Lib.connector import Connector  # import connector class


class Resistor(Component):
    """Model for lumped element resistor"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.filename = Path(__file__).parent /
"../images/lumped/resistor_60x30.png"

        self.in1, self.out = None, None
        self.conn_list = []

        self.create_image(self.filename)
        self.update_bbox()
        self.create_selector()
        self.create_connectors()  # Connector method
        self.set_connector_visibility()  # Connector method

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_selector()
        self.update_connectors()  # Connector method
        self.set_connector_visibility()  # Connector method
```

Comp_Lib/inductor.py

```
from pathlib import Path

from Comp_Lib.component import Component


class Inductor(Component):
    """Model for lumped element inductor"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.filename = Path(__file__).parent /
"../images/lumped/inductor_60x30.png"
        self.create_image(self.filename)
        self.update_bbox()
        self.create_selector()

        # Create connectors
        self.in1, self.out = None, None
        self.conn_list = []
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()
```

Comp_Lib/capacitor.py

```
from pathlib import Path

from Comp_Lib.component import Component


class Capacitor(Component):
    """Model for lumped element capacitor"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.filename = Path(__file__).parent /
"../images/lumped/capacitor_60x30.png"
        self.create_image(self.filename)
```

```
        self.update_bbox()
        self.create_selector()

        # Create connectors
        self.in1, self.out = None, None
        self.conn_list = []
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()
```

UI_Lib/canvas.py

```
import customtkinter as ctk

from UI_Lib.mouse import Mouse
from UI_Lib.grid import Grid


class Canvas(ctk.CTkCanvas):
    def __init__(self, parent):
        super().__init__(parent)

        self.led_color = "red"
        self.led_size = "large"
        self.grid = Grid(self, 10)
        self.mouse = Mouse(self)
        self.comp_list = []

        self.mouse.move_mouse_bind_events()

    def redraw(self):
        self.delete('grid_line')
        self.grid.draw()
        self.tag_lower("grid_line")
        for c in self.comp_list:
            c.update()
```

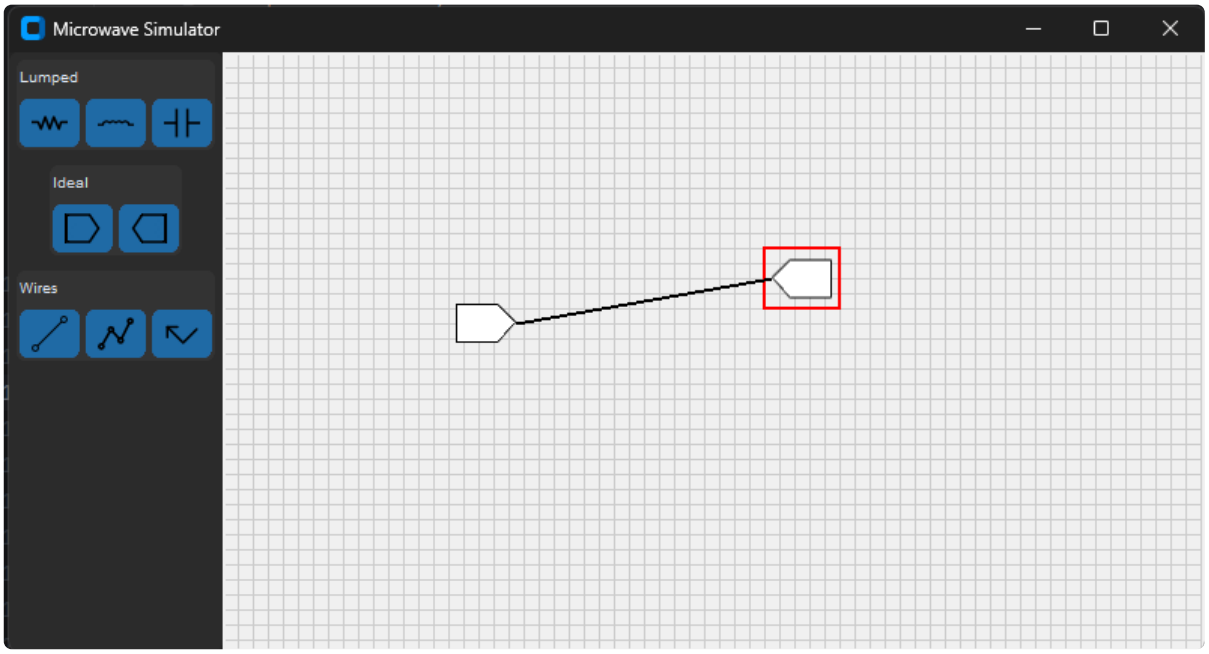
```
    def show_connectors(self):  # Show all component connectors
        for s in self.comp_list:
            s.is_drawing = True
        self.redraw()

    def hide_connectors(self):  # Hid all compoenent connector
        for s in self.comp_list:
            s.is_drawing = False
        self.redraw()

    def edit_shape(self, _event=None):
        pass
```

UI_Lib/mouse.py

```
from Helper_Lib import Point
from Wire_Lib import StraightWire, SegmentWire, ElbowWire
from Comp_Lib import Connection  # Import connection class


class Mouse:
    def __init__(self, canvas):
        self.canvas = canvas

        self.selected_comp = None
        self.current_wire_obj = None
        self.start = Point(0, 0)
        self.offset1 = Point(0, 0)
        self.offset2 = Point(0, 0)

        self.move_mouse_bind_events()

        . . .

    def draw_left_down(self, event):  # Added method for draw left down
        if self.current_wire_obj:
            # self.unselect_all()
            self.start.x = event.x
            self.start.y = event.y
            self.start.x, self.start.y =
self.canvas.grid.snap_to_grid(self.start.x, self.start.y)
```

```
            self.current_wire_obj.x1, self.current_wire_obj.y1 = self.start.x,
self.start.y
            self.current_wire_obj.x2, self.current_wire_obj.y2 = self.start.x,
self.start.y

            self.select_connector(self.current_wire_obj, "begin", self.start.x,
self.start.y)  # Check for connector hit

    def draw_left_drag(self, event):  # Added method for draw left drag
        if self.current_wire_obj:
            wire = self.current_wire_obj
            x, y = event.x, event.y
            x, y = self.canvas.grid.snap_to_grid(x, y)
            wire.x1, wire.y1 = self.start.x, self.start.y
            wire.x2, wire.y2 = x, y
            self.current_wire_obj.update()

    def draw_left_up(self, event):  # Added method for draw left up
        self.select_connector(self.current_wire_obj, "end", event.x, event.y)
# Check for connector hit
        self.canvas.hide_connectors()  # Hid connectors
        self.current_wire_obj = None
        self.move_mouse_bind_events()

        . . .

    def select_connector(self, wire_obj, wire_end, x, y):  # Added method to
see if line end hits a comp connector
        for comp in self.canvas.comp_list:
            if not (isinstance(comp, StraightWire) or isinstance(comp,
SegmentWire) or
                        isinstance(self.selected_comp, ElbowWire)):
                conn = comp.check_connector_hit(x, y)
                if conn:
                    if wire_end == "begin":
                        wire_obj.x1, wire_obj.y1 = conn.x, conn.y
                    elif wire_end == "end":
                        wire_obj.x2, wire_obj.y2 = conn.x, conn.y
                    a_conn = Connection(conn, self.current_wire_obj, wire_end)
                    comp.wire_list.append(a_conn)
                    self.canvas.redraw()
```

Port Classes



Comp_Lib/inport.py

```
from pathlib import Path

from Comp_Lib.component import Component
from Helper_Lib import Point
from Comp_Lib.connector import Connector


class Inport(Component):
    """Model for input port"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.filename = Path(__file__).parent / "../images/ports/in_port.png"

        self.in1, self.out = None, None
        self.conn_list = []

        self.create_image(self.filename)
        self.update_bbox()
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
```

```
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_connectors(self):  # Override base class method
        # Calculate position of connectors from current shape position and size
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w, h = x2 - x1, y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        self.out = Connector(self.canvas, "out", center.x + w / 2, center.y)

        # Update the connector list
        self.conn_list = [self.out]

    def update_connectors(self):  # Override base class method
        """Update the position of all connectors here"""
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w = x2 - x1
        h = y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        if self.angle == 0:
            self.out.x, self.out.y = center.x + w / 2, center.y
        elif self.angle == 90:
            self.out.x, self.out.y = center.x, center.y - h / 2
        elif self.angle == 180:
            self.out.x, self.out.y = center.x - w / 2, center.y
        elif self.angle == 270:
            self.out.x, self.out.y = center.x, center.y + h / 2

        for c in self.conn_list:
            c.update()

        self.move_connected_wires()
```

Comp_Lib/outport.py

```
from pathlib import Path

from Comp_Lib.component import Component
from Helper_Lib import Point
from Comp_Lib.connector import Connector
```

```
class Outport(Component):
    """Model for output port"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.filename = Path(__file__).parent / "../images/ports/out_port.png"

        self.in1, self.out = None, None
        self.conn_list = []

        self.create_image(self.filename)
        self.update_bbox()
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_connectors(self):  # Override base class method
        # Calculate position of connectors from current shape position and size
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w, h = x2 - x1, y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        self.in1 = Connector(self.canvas, "in1", center.x - w / 2, center.y)

        # Update the connector list
        self.conn_list = [self.in1]

    def update_connectors(self):  # Override base class method
        """Update the position of all connectors here"""
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w = x2 - x1
        h = y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        if self.angle == 0:
            self.in1.x, self.in1.y = center.x - w / 2, center.y
        elif self.angle == 90:
```

```
            self.in1.x, self.in1.y = center.x, center.y + h / 2
        elif self.angle == 180:
            self.in1.x, self.in1.y = center.x + w / 2, center.y
        elif self.angle == 270:
            self.in1.x, self.in1.y = center.x, center.y - h / 2

        for c in self.conn_list:
            c.update()

        self.move_connected_wires()
```

UI_Lib/ideal_button_frame.py

```
import customtkinter as ctk
from tktooltip import ToolTip
from pathlib import Path
from PIL import Image

from Comp_Lib import Inport, Outport


class IdealButtonFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.button_id = None

        self.button_list = [("inport", "../icons/inport.png"),
                            ("outport", "../icons/outport.png")]

        self.init_frame_widgets()

    def init_frame_widgets(self):
        frame_name_label = ctk.CTkLabel(self, text="Ideal", font=("Helvetica",
10), height=20)
        frame_name_label.grid(row=0, column=0, columnspan=3, sticky=ctk.W,
padx=2, pady=2)

        row_num, col_num = 1, 0
        for button in self.button_list:
            a_image = ctk.CTkImage(light_image=Image.open
                                   (Path(__file__).parent / button[1]),
```

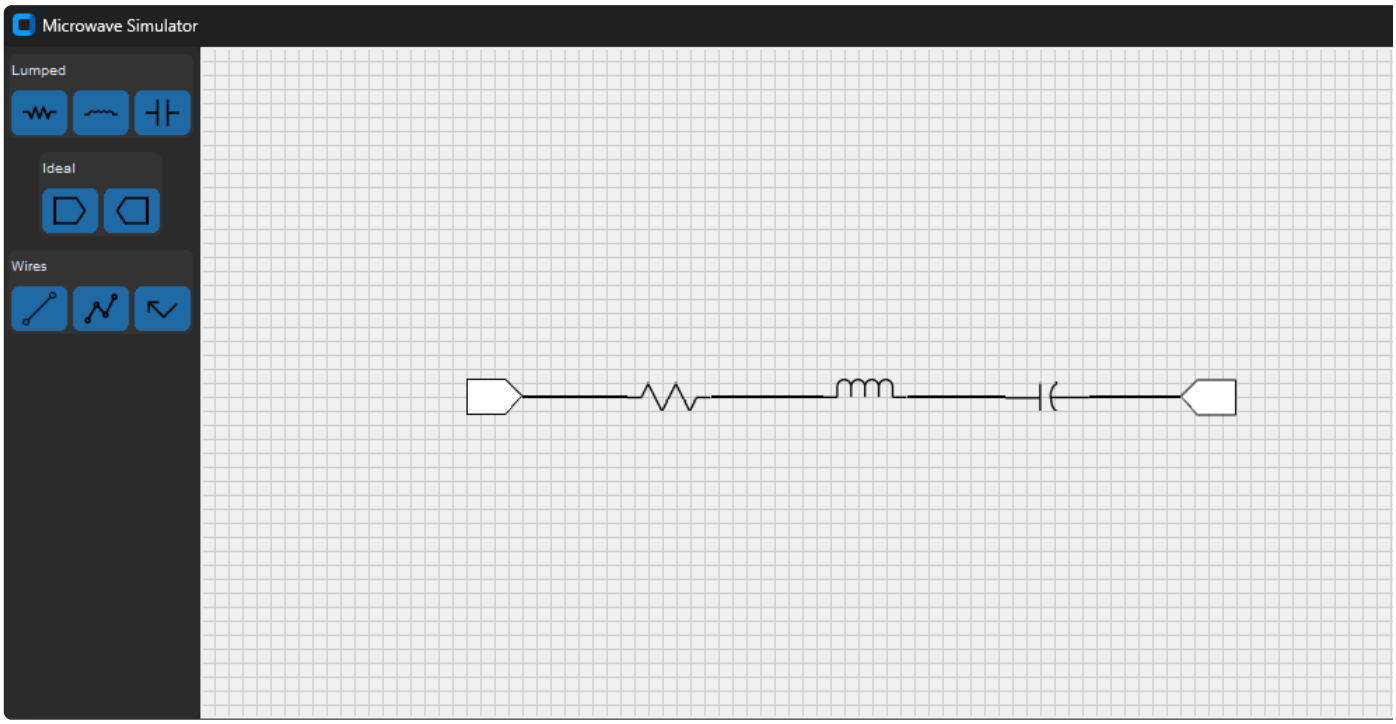
```
                                    dark_image=Image.open
                                    (Path(__file__).parent / button[1]),
                                    size=(24, 24))
            self.button_id = ctk.CTkButton(self, text="", image=a_image,
width=30,
                                           command=lambda
a_name=button[0]:self.create_events(a_name))
            self.button_id.grid(row=row_num, column=col_num, sticky=ctk.W,
padx=2, pady=2)
            ToolTip(self.button_id, msg=button[0])
            row_num, col_num = self.update_grid_numbers(row_num, col_num)

    def create_events(self, name):
        comp = None
        if name == "inport":
            comp = Inport(self.canvas, 100, 100)
        elif name == "outport":
            comp = Outport(self.canvas, 100, 100)
        self.canvas.comp_list.append(comp)
        self.canvas.redraw()
        self.canvas.mouse.move_mouse_bind_events()

    @staticmethod
    def update_grid_numbers(row, column):
        column += 1
        if column > 2:
            column = 0
            row += 1
        return row, column
```

UI_Lib/left_frame.py

```
import customtkinter as ctk
from UI_Lib import LumpButtonFrame, IdealButtonFrame, WireButtonFrame


class LeftFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.canvas = canvas

        self.comp_button_frame = LumpButtonFrame(self, self.canvas)
        self.comp_button_frame.pack(side=ctk.TOP, padx=5, pady=5)
```

```
        self.ideal_button_frame = IdealButtonFrame(self, self.canvas)  # Added
ideal button frame
        self.ideal_button_frame.pack(side=ctk.TOP, padx=5, pady=5)

        self.wire_button_frame = WireButtonFrame(self, self.canvas)
        self.wire_button_frame.pack(side=ctk.TOP, padx=5, pady=5)
```

RLC Circuit

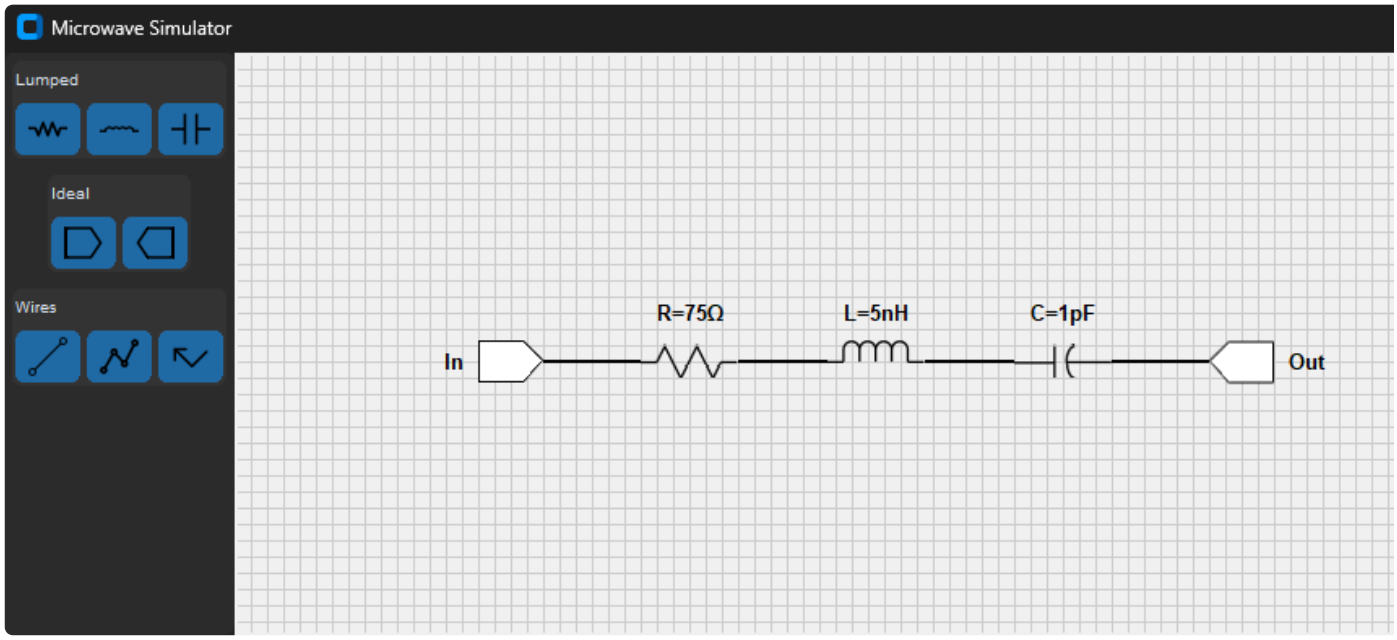


Circuit drawn without any additional code changes.

Objectives:

- Add text to components - show component values
- Save circuit
- Load circuit

Add Text to Components



Comp_Lib/inport.py

```
from pathlib import Path

from Comp_Lib.component import Component
from Helper_Lib import Point
from Comp_Lib.connector import Connector


class Inport(Component):
    """Model for input port"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.filename = Path(__file__).parent / "../images/ports/in_port.png"

        self.in1, self.out = None, None
        self.conn_list = []

        self.text = "In"  # Add text attribute
        self.text_id = None  # Id for text item

        self.create_image(self.filename)
        self.update_bbox()
        self.create_text()  # New method call to create text
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()
```

```
    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_text()  # New method call to update text
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_text(self):  # New method to create text
        self.text_id = self.canvas.create_text(self.x1-35, self.y1,
                                text=self.text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def update_text(self):  # New Method to update text
        self.canvas.coords(self.text_id, self.x1-35, self.y1)

        . . .
```

Comp_Lib/output.py

```
from pathlib import Path

from Comp_Lib.component import Component
from Helper_Lib import Point
from Comp_Lib.connector import Connector


class Outport(Component):
    """Model for output port"""
    def __init__(self, canvas, x1, y1):
        super().__init__(canvas, x1, y1)
        self.filename = Path(__file__).parent / "../images/ports/out_port.png"

        self.in1, self.out = None, None
        self.conn_list = []

        self.text = "Out"  # Add text attribute
        self.text_id = None  # Id for text item

        self.create_image(self.filename)
        self.update_bbox()
```

```
        self.create_text()  # New method call to create text
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_text()  # New method call to update text
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_text(self):  # New method to create text
        self.text_id = self.canvas.create_text(self.x1+40, self.y1,
                                text=self.text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def update_text(self):  # New Method to update text
        self.canvas.coords(self.text_id, self.x1+40, self.y1)

        . . .
```

Comp_Lib/resistor.py

```
from pathlib import Path

from Comp_Lib.component import Component


class Resistor(Component):
    """Model for lumped element resistor"""
    def __init__(self, canvas, x1, y1, resistance):  # Added resistance
argument
        super().__init__(canvas, x1, y1)
        self.resistance = resistance  # Added resistance attribute
        self.filename = Path(__file__).parent /
"../images/lumped/resistor_60x30.png"

        self.in1, self.out = None, None
        self.conn_list = []
```

```
        self.text = 'R=' + str(resistance) + '\u2126'  # Create resistor text
using resistance
        self.text_id = None  # Id for text item

        self.create_image(self.filename)
        self.update_bbox()
        self.create_text()  # New method call to create text
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_text()  # New method call to update text
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_text(self):  # New method to create text
        self.text_id = self.canvas.create_text(self.x1, self.y1-30,
                                text=self.text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def update_text(self):  # New Method to update text
        self.canvas.coords(self.text_id, self.x1, self.y1-30)
```

Comp_Lib/inductor.py

```
from pathlib import Path

from Comp_Lib.component import Component


class Inductor(Component):
    """Model for lumped element inductor"""
    def __init__(self, canvas, x1, y1, inductance):  # Added inductance
argument
        super().__init__(canvas, x1, y1)
        self.inductance = inductance  # Added inductance attribute
```

```
        self.filename = Path(__file__).parent /
"../images/lumped/inductor_60x30.png"

        self.in1, self.out = None, None
        self.conn_list = []

        self.text = 'L=' + str(inductance) + 'nH'  # Create inductor text using
inductance
        self.text_id = None  # Id for text item

        self.create_image(self.filename)
        self.update_bbox()
        self.create_text()  # New method call to create text
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_text()  # New method call to update text
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_text(self):  # New method to create text
        self.text_id = self.canvas.create_text(self.x1, self.y1-30,
                                text=self.text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def update_text(self):  # New method to update text
        self.canvas.coords(self.text_id, self.x1, self.y1-30)
```

Comp_Lib/capacitor.py

```
from pathlib import Path

from Comp_Lib.component import Component


class Capacitor(Component):
```

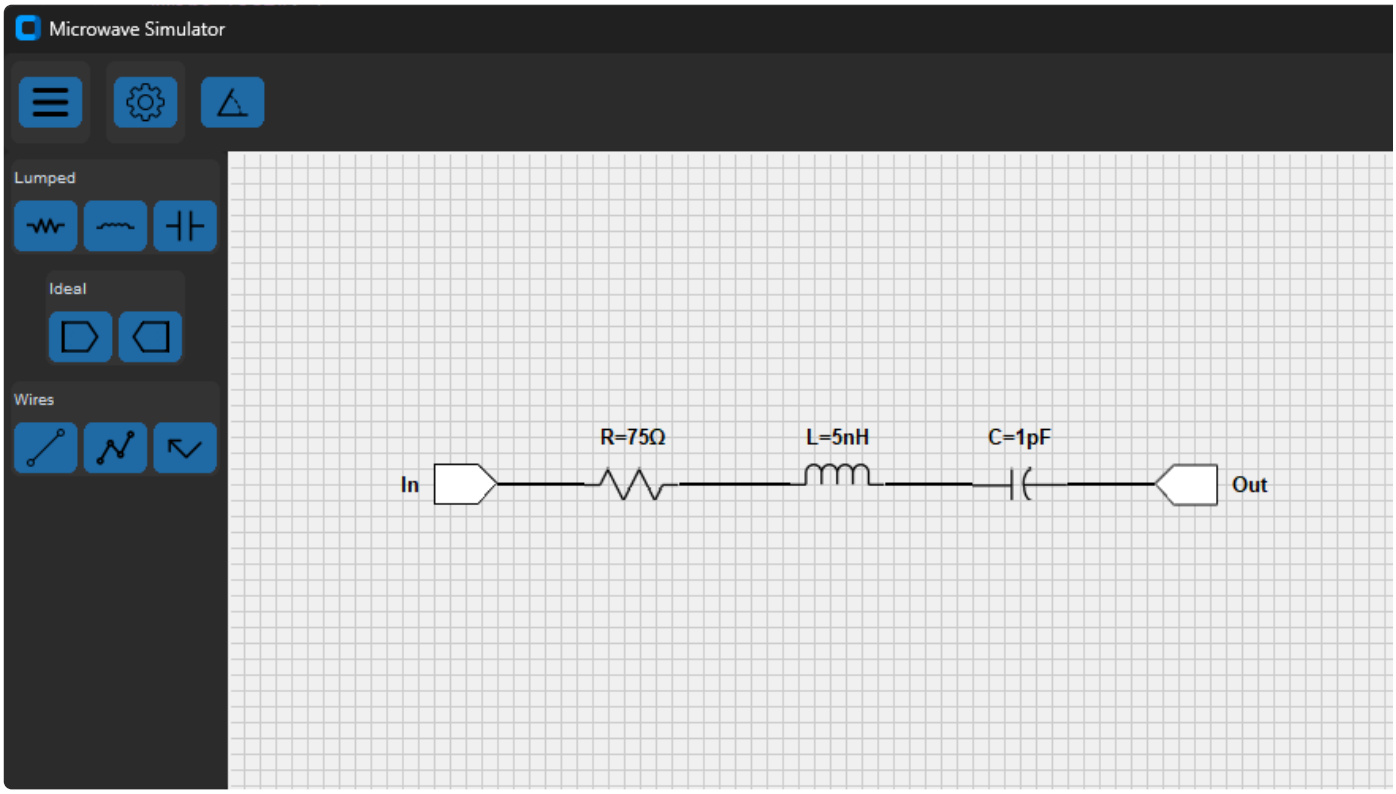
```
    """Model for lumped element capacitor"""
    def __init__(self, canvas, x1, y1, capacitance):  # Added capacitance
argument
        super().__init__(canvas, x1, y1)
        self.capacitance = capacitance  # Added capacitance attribute
        self.filename = Path(__file__).parent /
"../images/lumped/capacitor_60x30.png"

        self.in1, self.out = None, None
        self.conn_list = []

        self.text = 'C=' + str(capacitance) + 'pF'  # Create capacitor text
using capacitance
        self.text_id = None  # Id for text item

        self.create_image(self.filename)
        self.update_bbox()
        self.create_text()  # New method call to create text
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_text()  # New method call to update text
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_text(self):  # New method to create text
        self.text_id = self.canvas.create_text(self.x1, self.y1-30,
                                text=self.text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def update_text(self):  # New method to update text
        self.canvas.coords(self.text_id, self.x1, self.y1-30)
```

UI_Lib/left_button_frame.py

```
        . . .

    def create_events(self, name):
        gate = None
        if name == "resistor":
            gate = Resistor(self.canvas, 100, 100, 75)  # Init resistor with
resistance
        elif name == "inductor":
            gate = Inductor(self.canvas, 100, 100, 5)  # Init inductor with
inductance
        elif name == "capacitor":
            gate = Capacitor(self.canvas, 100, 100, 1)  # Init capacitor with
capacitance
        self.canvas.comp_list.append(gate)
        self.canvas.redraw()
        self.canvas.mouse.move_mouse_bind_events()

        . . .
```

File Save & Load

Screen after file save & load



UI_Lib/file_menu_frame.py

```
import customtkinter as ctk
from tkinter import filedialog as fd
from pathlib import Path
import json
from PIL import Image

from Comp_Lib import Resistor, Inductor, Capacitor
from Comp_Lib import Inport, Outport, Connection
from Wire_Lib import StraightWire, SegmentWire, ElbowWire


class MyEncoder(json.JSONEncoder):
    def default(self, o):
        if hasattr(o, "reprJson"):
            return o.reprJson()
        else:
            return super().default(o)


class JSONDCoder(json.JSONDecoder):
    def __init__(self):
        json.JSONDecoder.__init__(self, object_hook=JSONDCoder.from_dict)

    @staticmethod
    def from_dict(_d):
        return _d


class FileMenuFrame(ctk.CTkFrame):
    def __init__(self, window, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.obj_type_dict = {'resistor': Resistor,
                              'inductor': Inductor,
                              'capacitor': Capacitor,
                              'inport': Inport,
                              'outport': Outport,
                              'straight': StraightWire,
                              'segment': SegmentWire,
                              'elbow': ElbowWire}

        self.menu_on = False
```

```
        self.menu_frame = ctk.CTkFrame(window, height=100, bg_color="white")

        new_btn = ctk.CTkButton(self.menu_frame, text="New", width=150,
command=self.new_diagram)
        new_btn.pack(pady=5)

        open_btn = ctk.CTkButton(self.menu_frame, text="Open", width=150,
command=self.load_diagram)
        open_btn.pack(pady=5)

        save_btn = ctk.CTkButton(self.menu_frame, text="Save", width=150,
command=self.save_diagram)
        save_btn.pack(pady=5)

        exit_btn = ctk.CTkButton(self.menu_frame, text="Exit", width=150,
command=window.destroy)
        exit_btn.pack(pady=5)

        my_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/hamburger_menu.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/hamburger_menu.png"),
                                size=(24, 24))

        button = ctk.CTkButton(self, text="", image=my_image, width=30,
command=self.show_menu)
        button.pack(side=ctk.LEFT, padx=5, pady=10)

    def new_diagram(self):
        self.canvas.delete("all")
        self.canvas.comp_list = []

    def load_diagram(self):
        try:
            filetypes = (('json files', '*.json'), ('All files', '*.*'))
            f = fd.askopenfilename(filetypes=filetypes, initialdir="./")
            with open(f) as file:
                d = json.load(file)
                # print(d)

                for item in d:
                    if item['type'] == "straight":
                        wire = StraightWire(self.canvas, x1=item['x1'],
```

```
y1=item['y1'], x2=item['x2'], y2=item['y2'])
                        self.canvas.comp_list.append(wire)
                    elif item['type'] == "segment":
                        wire = SegmentWire(self.canvas, x1=item['x1'],
y1=item['y1'], x2=item['x2'], y2=item['y2'])
                        self.canvas.comp_list.append(wire)
                    elif item['type'] == "elbow":
                        wire = ElbowWire(self.canvas, x1=item['x1'],
y1=item['y1'], x2=item['x2'], y2=item['y2'])
                        self.canvas.comp_list.append(wire)
                    elif item['type'] == "resistor":
                        resistor = Resistor(self.canvas, x1=item['x1'],
y1=item['y1'], resistance=item['resistance'])
                        resistor.angle = item['angle']
                        self.canvas.comp_list.append(resistor)
                    elif item['type'] == "inductor":
                        inductor = Inductor(self.canvas, x1=item['x1'],
y1=item['y1'], inductance=item['inductance'])
                        inductor.angle = item['angle']
                        self.canvas.comp_list.append(inductor)
                    elif item['type'] == "capacitor":
                        capacitor = Capacitor(self.canvas, x1=item['x1'],
y1=item['y1'],
                                              capacitance=item['capacitance'])
                        capacitor.angle = item['angle']
                        self.canvas.comp_list.append(capacitor)
                    elif item['type'] == "inport":
                        inport = Inport(self.canvas, x1=item['x1'],
y1=item['y1'])
                        inport.angle = item['angle']
                        self.canvas.comp_list.append(inport)
                    elif item['type'] == "outport":
                        outport = Outport(self.canvas, x1=item['x1'],
y1=item['y1'])
                        outport.angle = item['angle']
                        self.canvas.comp_list.append(outport)

                # Add connections
                for item in d:
                    if item['type'] == "resistor":
                        wire_list = item['wire_list']
                        for wire_item in wire_list:
                            x1 = wire_item['wire_obj']['x1']
                            y1 = wire_item['wire_obj']['y1']
                            x2 = wire_item['wire_obj']['x2']
                            y2 = wire_item['wire_obj']['y2']
```

```
                            # Test to see if wire obj matches wire coordinates
                            if x1 == wire.x1 and y1 == wire.y1 and x2 ==
wire.x2 and y2 == wire.y2:
                                conn = Connection(resistor, wire,
wire_item['wire_end'])
                                resistor.wire_list.append(conn)

                    elif item['type'] == "inductor":
                        wire_list = item['wire_list']
                        for wire_item in wire_list:
                            x1 = wire_item['wire_obj']['x1']
                            y1 = wire_item['wire_obj']['y1']
                            x2 = wire_item['wire_obj']['x2']
                            y2 = wire_item['wire_obj']['y2']
                            # Test to see if wire obj matches wire coordinates
                            if x1 == wire.x1 and y1 == wire.y1 and x2 ==
wire.x2 and y2 == wire.y2:
                                conn = Connection(inductor, wire,
wire_item['wire_end'])
                                inductor.wire_list.append(conn)

                    elif item['type'] == "capacitor":
                        wire_list = item['wire_list']
                        for wire_item in wire_list:
                            x1 = wire_item['wire_obj']['x1']
                            y1 = wire_item['wire_obj']['y1']
                            x2 = wire_item['wire_obj']['x2']
                            y2 = wire_item['wire_obj']['y2']
                            # Test to see if wire obj matches wire coordinates
                            if x1 == wire.x1 and y1 == wire.y1 and x2 ==
wire.x2 and y2 == wire.y2:
                                conn = Connection(capacitor, wire,
wire_item['wire_end'])
                                capacitor.wire_list.append(conn)

                    elif item['type'] == "inport":
                        wire_list = item['wire_list']
                        for wire_item in wire_list:
                            x1 = wire_item['wire_obj']['x1']
                            y1 = wire_item['wire_obj']['y1']
                            x2 = wire_item['wire_obj']['x2']
                            y2 = wire_item['wire_obj']['y2']
                            # Test to see if wire obj matches wire coordinates
                            if x1 == wire.x1 and y1 == wire.y1 and x2 ==
wire.x2 and y2 == wire.y2:
                                conn = Connection(inport, wire,
```

```
wire_item['wire_end'])
                                inport.wire_list.append(conn)

                    elif item['type'] == "outport":
                        wire_list = item['wire_list']
                        for wire_item in wire_list:
                            x1 = wire_item['wire_obj']['x1']
                            y1 = wire_item['wire_obj']['y1']
                            x2 = wire_item['wire_obj']['x2']
                            y2 = wire_item['wire_obj']['y2']
                            # Test to see if wire obj matches wire coordinates
                            if x1 == wire.x1 and y1 == wire.y1 and x2 ==
wire.x2 and y2 == wire.y2:
                                conn = Connection(outport, wire,
wire_item['wire_end'])
                                outport.wire_list.append(conn)

        except FileNotFoundError:
            with open('untitled.canvas', 'w') as _file:
                pass
            self.canvas.comp_list = []

    def save_diagram(self):
        filetypes = (('json files', '*.json'), ('All files', '*.*'))
        f = fd.asksaveasfilename(filetypes=filetypes, initialdir="./")
        with open(f, 'w') as file:
            file.write(json.dumps(self.canvas.comp_list, cls=MyEncoder,
indent=4))

    def show_menu(self):
        if not self.menu_on:
            self.menu_frame.place(x=15, y=60)
            self.menu_frame.tkraise()
            self.menu_on = True
        else:
            self.menu_frame.place_forget()
            self.menu_on = False
```

Add `reprJson()` to each component file for the JSON encoder
Comp_Lib/inport.py and outport.py

```
        . . .
```

```
         def reprJson(self):
        return dict(type=self.type, x1=self.x1, y1=self.y1, angle=self.angle,
wire_list=self.wire_list)
```

Comp_Lib/resistor.py

```
        . . .

            def reprJson(self):
        return dict(type=self.type, x1=self.x1, y1=self.y1, angle=self.angle,
resistance=self.resistance,
                    wire_list=self.wire_list)
```

Comp_Lib/inductor.py

```
        . . .

            def reprJson(self):
        return dict(type=self.type, x1=self.x1, y1=self.y1, angle=self.angle,
inductance=self.inductance,
                    wire_list=self.wire_list)
```

Comp_Lib/capacitor.py

```
        . . .

            def reprJson(self):
        return dict(type=self.type, x1=self.x1, y1=self.y1, angle=self.angle,
capacitance=self.capacitance,
                    wire_list=self.wire_list)
```

Wire_lib/straight_wire.py & segment_wire.py & elbow_wire.py

```
        . . .

            def reprJson(self):
        return dict(type=self.type, x1=self.x1, y1=self.y1, x2=self.x2,
y2=self.y2)
```

`rlc.json` - file saved


```
[
    {
        "type": "inport",
        "x1": 150,
        "y1": 210,
        "angle": 0,
        "wire_list": [
            {
                "type": "connection",
                "wire_obj": {
                    "type": "straight",
                    "x1": 170.0,
                    "y1": 210.0,
                    "x2": 225.0,
                    "y2": 210.0
                },
                "wire_end": "begin"
            }
        ]
    },
    {
        "type": "resistor",
        "x1": 255,
        "y1": 210,
        "angle": 0,
        "resistance": 75,
        "wire_list": [
            {
                "type": "connection",
                "wire_obj": {
                    "type": "straight",
                    "x1": 170.0,
                    "y1": 210.0,
                    "x2": 225.0,
                    "y2": 210.0
```

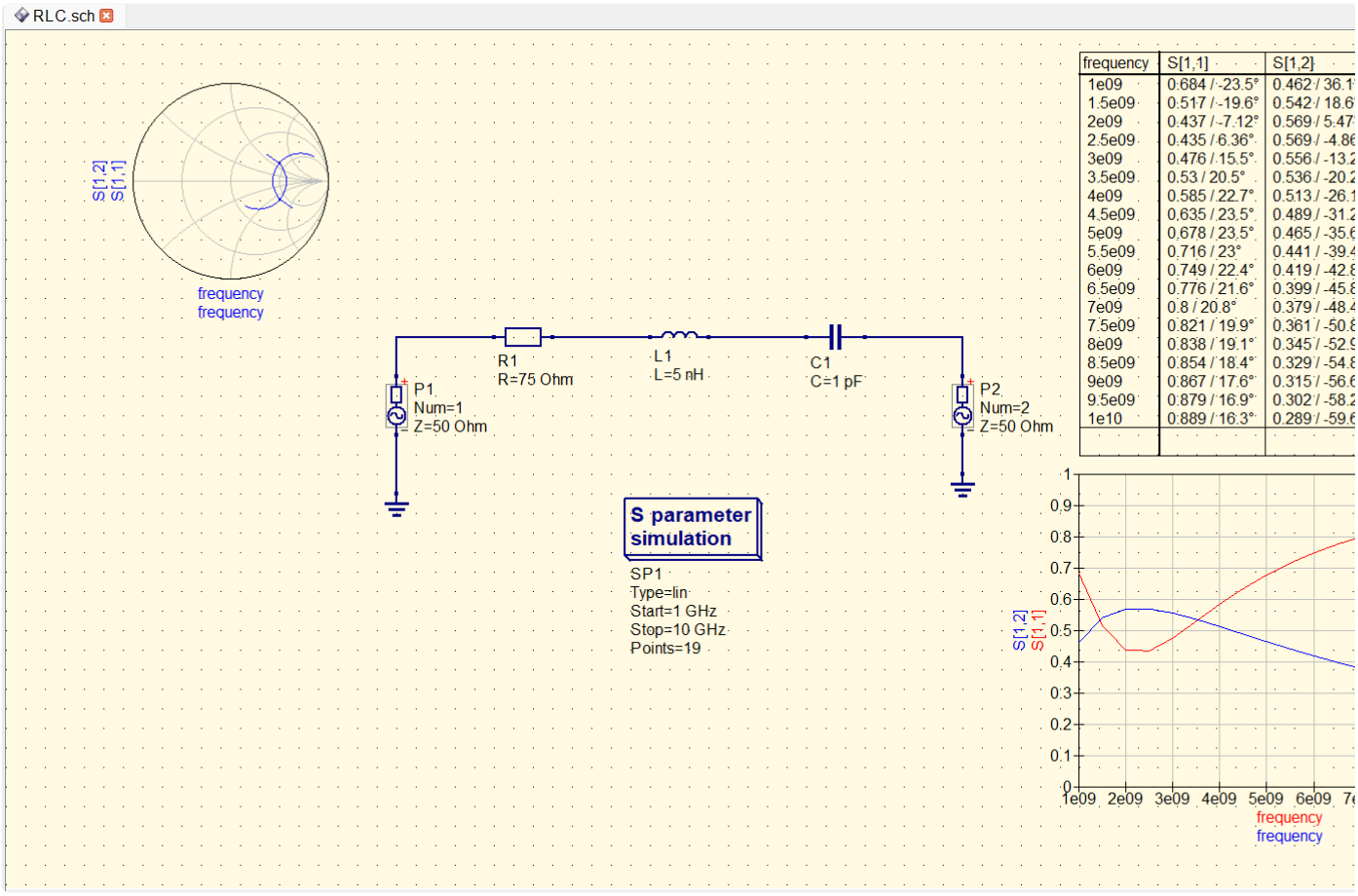
```
                },
                "wire_end": "end"
            },
            {
                "type": "connection",
                "wire_obj": {
                    "type": "straight",
                    "x1": 285.0,
                    "y1": 210.0,
                    "x2": 355.0,
                    "y2": 210.0
                },
                "wire_end": "begin"
            }
        ]
    },
    {
        "type": "inductor",
        "x1": 385,
        "y1": 210,
        "angle": 0,
        "inductance": 5,
        "wire_list": [
            {
                "type": "connection",
                "wire_obj": {
                    "type": "straight",
                    "x1": 285.0,
                    "y1": 210.0,
                    "x2": 355.0,
                    "y2": 210.0
                },
                "wire_end": "end"
            },
            {
                "type": "connection",
                "wire_obj": {
                    "type": "straight",
                    "x1": 415.0,
                    "y1": 210.0,
                    "x2": 470.0,
                    "y2": 210.0
                },
                "wire_end": "begin"
            }
        ]
```

```
    },
    {
        "type": "capacitor",
        "x1": 500,
        "y1": 210,
        "angle": 0,
        "capacitance": 1,
        "wire_list": [
            {
                "type": "connection",
                "wire_obj": {
                    "type": "straight",
                    "x1": 415.0,
                    "y1": 210.0,
                    "x2": 470.0,
                    "y2": 210.0
                },
                "wire_end": "end"
            },
            {
                "type": "connection",
                "wire_obj": {
                    "type": "straight",
                    "x1": 530.0,
                    "y1": 210.0,
                    "x2": 585.0,
                    "y2": 210.0
                },
                "wire_end": "begin"
            }
        ]
    },
    {
        "type": "outport",
        "x1": 605,
        "y1": 210,
        "angle": 0,
        "wire_list": [
            {
                "type": "connection",
                "wire_obj": {
                    "type": "straight",
                    "x1": 530.0,
                    "y1": 210.0,
                    "x2": 585.0,
                    "y2": 210.0
```

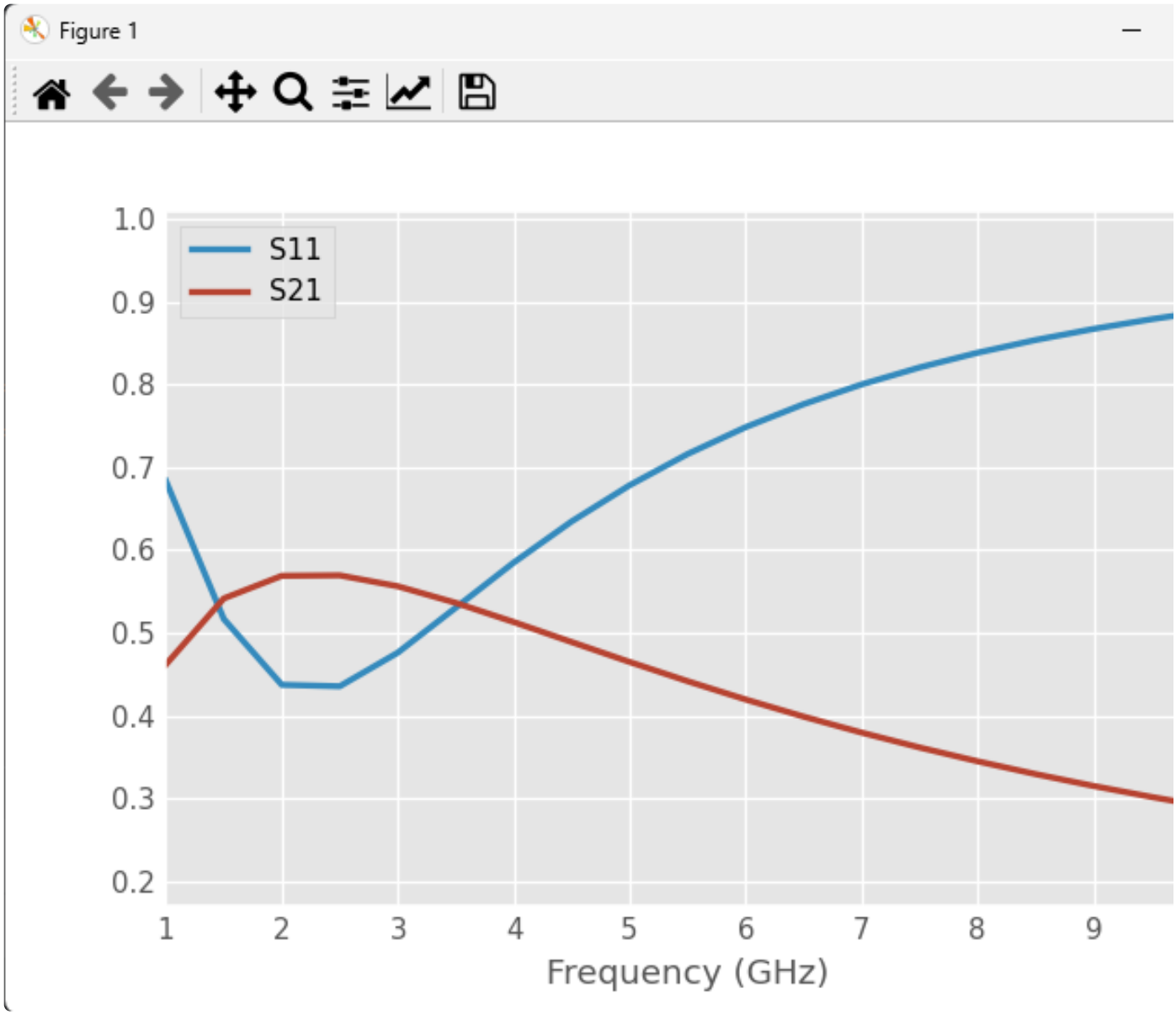
```
                },
                "wire_end": "end"
            }
        ]
    },
    {
        "type": "straight",
        "x1": 170.0,
        "y1": 210.0,
        "x2": 225.0,
        "y2": 210.0
    },
    {
        "type": "straight",
        "x1": 285.0,
        "y1": 210.0,
        "x2": 355.0,
        "y2": 210.0
    },
    {
        "type": "straight",
        "x1": 415.0,
        "y1": 210.0,
        "x2": 470.0,
        "y2": 210.0
    },
    {
        "type": "straight",
        "x1": 530.0,
        "y1": 210.0,
        "x2": 585.0,
        "y2": 210.0
    }
]
```


RLC Circuit Analysis

QUCS Analysis



Scikit-RF Analysis



Console output

```
D:\EETools\MicrowaveSimulator\venv\Scripts\python.exe
D:\EETools\MicrowaveSimulator\Sandbox\RLC_example.py
[[[0.68368911 0.46154922]
  [0.46154922 0.68368911]]

 [[0.51666132 0.54150201]
  [0.54150201 0.51666132]]

 [[0.43712731 0.56883028]
  [0.56883028 0.43712731]]
```

```
[[0.43535197 0.56937463]
 [0.56937463 0.43535197]]

[[0.47595789 0.55622445]
 [0.55622445 0.47595789]]

[[0.53033858 0.5361869 ]
 [0.5361869  0.53033858]]

[[0.58489041 0.51299248]
 [0.51299248 0.58489041]]

[[0.63466307 0.48875466]
 [0.48875466 0.63466307]]

[[0.67836989 0.46467808]
 [0.46467808 0.67836989]]

[[0.71612899 0.4414337 ]
 [0.4414337  0.71612899]]

[[0.7485435  0.41937221]
 [0.41937221 0.7485435 ]]

[[0.77633296 0.39865129]
 [0.39865129 0.77633296]]

[[0.80018956 0.37931342]
 [0.37931342 0.80018956]]

[[0.82072802 0.36133393]
 [0.36133393 0.82072802]]

[[0.83847445 0.34465089]
 [0.34465089 0.83847445]]

[[0.85387096 0.32918346]
 [0.32918346 0.85387096]]

[[0.86728573 0.31484312]
```

```
  [0.31484312 0.86728573]]

 [[0.87902421 0.30154034]
  [0.30154034 0.87902421]]

 [[0.88933967 0.28918848]
  [0.28918848 0.88933967]]]

Process finished with exit code 0
```

Sandbox/RLC_example.py

```
import skrf as rf
import matplotlib.pyplot as plt
rf.stylely()

freq = rf.Frequency(start=1.0, stop=10.0, unit='GHz', npoints=19)
tl_media = rf.DefinedGammaZ0(freq, z0=50)
R1 = rf.Circuit.SeriesImpedance(frequency=freq, name='R1', z0=50,
                                Z=75.0)
L1 = rf.Circuit.SeriesImpedance(frequency=freq, name='L1', z0=50,
                                Z=1j*freq.w*5e-9)
C1 = rf.Circuit.SeriesImpedance(frequency=freq, name='C1', z0=50,
                                Z=1/(1j*freq.w*1e-12))

port1 = rf.Circuit.Port(freq, name='port1', z0=50)
port2 = rf.Circuit.Port(freq, name='port2', z0=50)

cnx = [
    [(port1, 0), (R1, 0)],  # port1 output to R1 input
    [(R1, 1), (L1, 0)],     # R1 output to L1 input
    [(L1, 1), (C1, 0)],     # L1 output to C1 input
    [(C1, 1), (port2, 0)]   # C1 output to port2 input
]
ckt = rf.Circuit(cnx)
ntw = ckt.network

print(ntw.s_mag)

ntw.plot_s_mag(m=0, n=0, lw=2, logx=False)
ntw.plot_s_mag(m=1, n=0, lw=2, logx=False)
```

```
plt.show()
```

Basically, we need to convert the circuit diagram to a circuit list with connections and run the analysis.

Component List Analysis

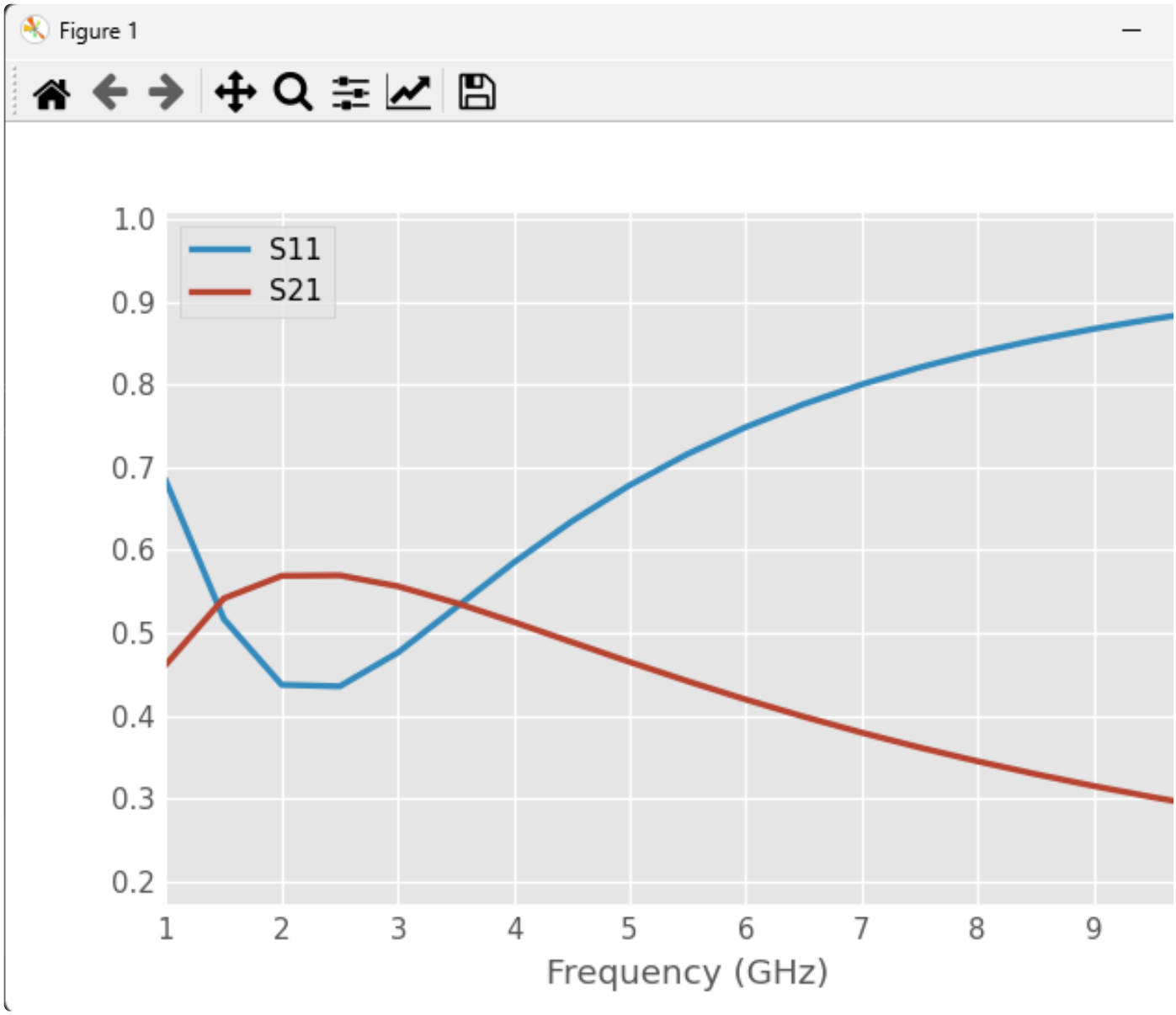
Sandbox/RLC_component_list_processor.py

```
import skrf as rf
import matplotlib.pyplot as plt
rf.stylely()


class Resistor:
    def __init__(self, x, y, resistance):
        self.type = 'resistor'


class Inductor:
    def __init__(self, x, y, inductance):
        self.type = 'inductor'


class Capacitor:
    def __init__(self, x, y, capacitance):
        self.type = 'capacitor'


class Inport:
    def __init__(self, x, y):
        self.type = 'inport'


class Outport:
    def __init__(self, x, y):
        self.type = 'outport'


class StraightWire:
```

```
    def __init__(self, x1, y1, x2, y2):
        self.type = 'wire'


# Create component list
def create_comp_list():
    # Components
    comp_list = []
    Res = Resistor(100, 100, 75)
    comp_list.append(Res)
    Ind = Inductor(100, 100, 5)
    comp_list.append(Ind)
    Cap = Capacitor(100, 100, 1)
    comp_list.append(Cap)
    inp = Inport(100, 100)
    comp_list.append(inp)
    outp = Outport(100, 100)
    comp_list.append(outp)

    # Wires
    wire1 = StraightWire(100, 100, 200, 200)
    wire2 = StraightWire(100, 100, 200, 200)
    wire3 = StraightWire(100, 100, 200, 200)
    wire4 = StraightWire(100, 100, 200, 200)
    comp_list.append(wire1)
    comp_list.append(wire2)
    comp_list.append(wire3)
    comp_list.append(wire4)

    # Create wire list
    wire_list = [
        [(inp, 'out'), (Res, 'in1')],
        [(Res, 'out'), (Ind, 'in1')],
        [(Ind, 'out'), (Cap, 'in1')],
        [(Cap, 'out'), (outp, 'in1')]
    ]

    return comp_list, wire_list


def analyze(component_list, wire_list):
    cnx = []
    R1, L1, C1, port1, port2 = None, None, None, None, None
    freq = rf.Frequency(start=1.0, stop=10.0, unit='GHz', npoints=19)

    # Convert graphical components to microwave components
```

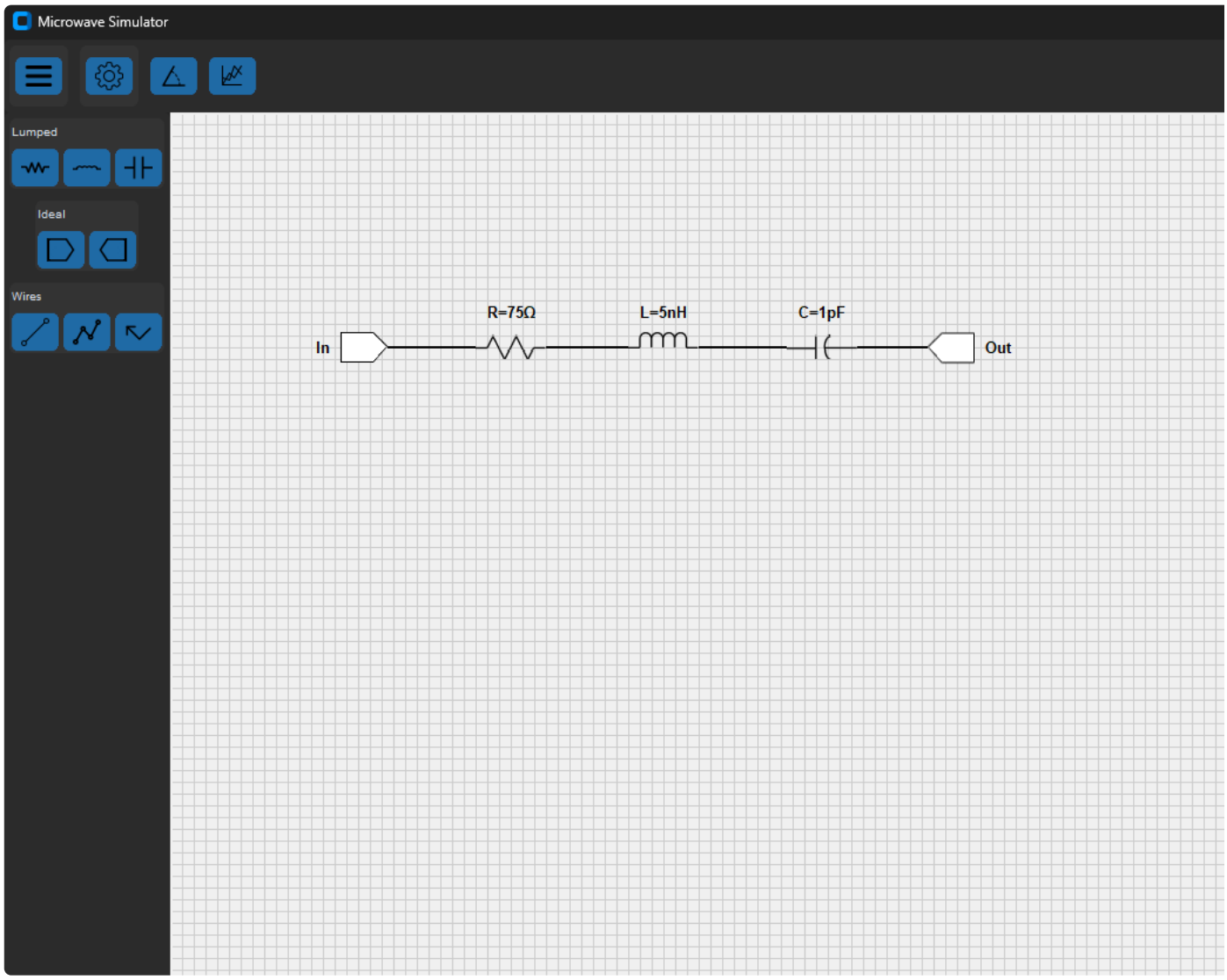
```
    for comp in component_list:
        if isinstance(comp, Resistor):
            R1 = rf.Circuit.SeriesImpedance(frequency=freq, name='R1', z0=50,
Z=75.0)
        elif isinstance(comp, Inductor):
            L1 = rf.Circuit.SeriesImpedance(frequency=freq, name='L1', z0=50,
Z=1j * freq.w * 5e-9)
        elif isinstance(comp, Capacitor):
            C1 = rf.Circuit.SeriesImpedance(frequency=freq, name='C1', z0=50,
Z=1 / (1j * freq.w * 1e-12))
        elif isinstance(comp, Inport):
            port1 = rf.Circuit.Port(freq, name='port1', z0=50)
        elif isinstance(comp, Outport):
            port2 = rf.Circuit.Port(freq, name='port2', z0=50)

    comp_dict = {
        'resistor': R1,
        'inductor': L1,
        'capacitor': C1,
        'inport': port1,
        'outport': port2,
        'wire': None
    }

    wire_end_dict = {
        'in1': 0,
        'out': 1
    }

    # Convert wire list to cnx list
    print(wire_list)
    for wire in wire_list:
        # Parse wire list
        comp1 = wire[0][0].type
        if comp1 == "inport":
            end1 = 'in1'  # Hack for Scikit-RF input port which labels the
output port as 0, I labeled them 'out'
        else:
            end1 = wire[0][1]
        comp2 = wire[1][0].type
        end2 = wire[1][1]

        # Convert to cnx list
        a_conx = [(comp_dict[comp1], wire_end_dict[end1]), (comp_dict[comp2],
wire_end_dict[end2])]
        cnx.append(a_conx)
```

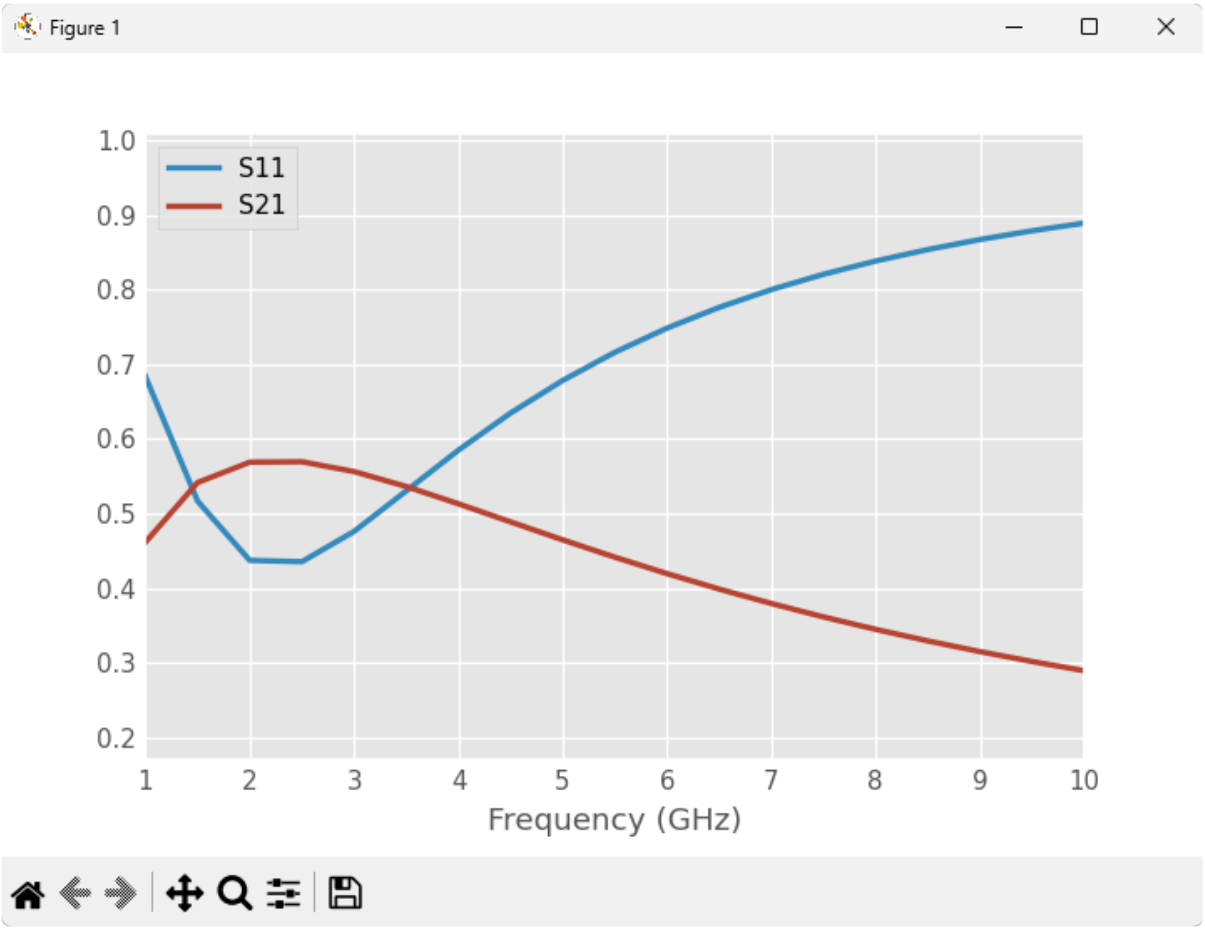
```
    # cnx = [
    #     [(port1, 0), (R1, 0)],  # port1 output to R1 input
    #     [(R1, 1), (L1, 0)],  # R1 output to L1 input
    #     [(L1, 1), (C1, 0)],  # L1 output to C1 input
    #     [(C1, 1), (port2, 0)]  # C1 output to port2 input
    # ]

    ckt = rf.Circuit(cnx)
    ntw = ckt.network

    print(ntw.s_mag)

    ntw.plot_s_mag(m=0, n=0, lw=2, logx=False)
    ntw.plot_s_mag(m=1, n=0, lw=2, logx=False)
    plt.show()


# Analyze the circuit
a_comp_list, a_wire_list = create_comp_list()
analyze(a_comp_list, a_wire_list)
```



The key here is to use the component type attribute (used for file save/load also) to create a dictionary that maps graphical component classes to microwave components. I also used a dictionary to map connection names ('in1', 'out') to 0, 1 respectively, which is the notation used for Scikit-RF component ports.

Analysis


Microwave Simulator
Lumped
Ideal
Wires
R=75Ω
L=5nH
C=1pF
In
Out



Console output

```
[[[0.68368911 0.46154922]
  [0.46154922 0.68368911]]

 [[0.51666132 0.54150201]
  [0.54150201 0.51666132]]

 [[0.43712731 0.56883028]
  [0.56883028 0.43712731]]

 [[0.43535197 0.56937463]
  [0.56937463 0.43535197]]

 [[0.47595789 0.55622445]
  [0.55622445 0.47595789]]

 [[0.53033858 0.5361869 ]
  [0.5361869  0.53033858]]
```

```
[[0.58489041 0.51299248]
 [0.51299248 0.58489041]]

[[0.63466307 0.48875466]
 [0.48875466 0.63466307]]

[[0.67836989 0.46467808]
 [0.46467808 0.67836989]]

[[0.71612899 0.4414337 ]
 [0.4414337  0.71612899]]

[[0.7485435  0.41937221]
 [0.41937221 0.7485435 ]]

[[0.77633296 0.39865129]
 [0.39865129 0.77633296]]

[[0.80018956 0.37931342]
 [0.37931342 0.80018956]]

[[0.82072802 0.36133393]
 [0.36133393 0.82072802]]

[[0.83847445 0.34465089]
 [0.34465089 0.83847445]]

[[0.85387096 0.32918346]
 [0.32918346 0.85387096]]

[[0.86728573 0.31484312]
 [0.31484312 0.86728573]]

[[0.87902421 0.30154034]
 [0.30154034 0.87902421]]

[[0.88933967 0.28918848]
 [0.28918848 0.88933967]]]
```

UI_Lib/top_frame.py

```
import customtkinter as ctk
from tktooltip import ToolTip
from pathlib import Path
from PIL import Image

import skrf as rf  # Added new import
import matplotlib.pyplot as plt  # Added new import
rf.stylely()  # Added

from UI_Lib.file_menu_frame import FileMenuFrame
from UI_Lib.settings_frame import SettingsFrame
from UI_Lib.help_frame import HelpFrame

from Comp_Lib import Resistor, Inductor, Capacitor, Inport, Outport  # Added
new import


class TopFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas

        # Add Top Frame widget here
        file_frame = FileMenuFrame(self.parent, self, self.canvas)
        file_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        settings_frame = SettingsFrame(self.parent, self, self.canvas)
        settings_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        help_frame = HelpFrame(self.parent, self, self.canvas)
        help_frame.pack(side=ctk.RIGHT, padx=5, pady=5)

        a_image = ctk.CTkImage(light_image=Image.open(Path(__file__).parent /
"../icons/angle.png"),
                               dark_image=Image.open(Path(__file__).parent /
"../icons/angle.png"),
                               size=(24, 24))
        self.button_id = ctk.CTkButton(self, text="", image=a_image, width=30,
command=self.rotate_comp)
        self.button_id.pack(side=ctk.LEFT, padx=5, pady=5)
        ToolTip(self.button_id, msg="Rotate selected component")
```

```
        a_image = ctk.CTkImage(light_image=Image.open(Path(__file__).parent /
"../icons/analyze.png"),
                               dark_image=Image.open(Path(__file__).parent /
"../icons/analyze.png"),
                               size=(24, 24))
        self.button_id = ctk.CTkButton(self, text="", image=a_image, width=30,
command=self.analyze_circuit)  # Call to analyze method
        self.button_id.pack(side=ctk.LEFT, padx=5, pady=5)
        ToolTip(self.button_id, msg="Analyze circuit")

    def rotate_comp(self):
        self.parent.rotate_comp(_event=None)

    def analyze_circuit(self):  # New analyze method
        comp_list = self.canvas.comp_list
        wire_list = self.canvas.wire_list
        cnx = []

        res, ind, cap, inport, outport = None, None, None, None, None
        freq = rf.Frequency(start=1.0, stop=10.0, unit='GHz', npoints=19)

        # Convert graphical components to microwave components
        for comp in comp_list:
            if isinstance(comp, Resistor):
                res = rf.Circuit.SeriesImpedance(frequency=freq, name='res',
z0=50, Z=75.0)
            elif isinstance(comp, Inductor):
                ind = rf.Circuit.SeriesImpedance(frequency=freq, name='ind',
z0=50, Z=1j * freq.w * 5e-9)
            elif isinstance(comp, Capacitor):
                cap = rf.Circuit.SeriesImpedance(frequency=freq, name='cap',
z0=50, Z=1 / (1j * freq.w * 1e-12))
            elif isinstance(comp, Inport):
                inport = rf.Circuit.Port(freq, name='inport', z0=50)
            elif isinstance(comp, Outport):
                outport = rf.Circuit.Port(freq, name='outport', z0=50)

        comp_dict = {
            'resistor': res,
            'inductor': ind,
            'capacitor': cap,
            'inport': inport,
            'outport': outport,
            'wire': None
        }
```

```
        wire_end_dict = {
            'in1': 0,
            'out': 1
        }

        # Convert wire list to cnx list
        print(wire_list)
        for wire in wire_list:
            # Parse wire list
            comp1 = wire[0][0].type
            if comp1 == "inport":
                end1 = 'in1'  # Hack for Scikit-RF input port which labels the
output port as 0, I labeled them 'out'
            else:
                end1 = wire[0][1]
            comp2 = wire[1][0].type
            end2 = wire[1][1]

            # Convert to cnx list
            a_conx = [(comp_dict[comp1], wire_end_dict[end1]),
(comp_dict[comp2], wire_end_dict[end2])]
            cnx.append(a_conx)

        ckt = rf.Circuit(cnx)
        ntw = ckt.network

        print(ntw.s_mag)  # Print S-parameters to console

        ntw.plot_s_mag(m=0, n=0, lw=2, logx=False)
        ntw.plot_s_mag(m=1, n=0, lw=2, logx=False)
        plt.show()  # Display S-parmenter plot
```

UI_Lib/canvas.py

```
import customtkinter as ctk

from UI_Lib.mouse import Mouse
from UI_Lib.grid import Grid


class Canvas(ctk.CTkCanvas):
    def __init__(self, parent):
        super().__init__(parent)
```

```
        self.led_color = "red"
        self.led_size = "large"
        self.grid = Grid(self, 10)
        self.mouse = Mouse(self)
        self.comp_list = []
        self.wire_list = []  # Added new wire list

        self.mouse.move_mouse_bind_events()

        . . .
```

Wire_Lib/wire.py

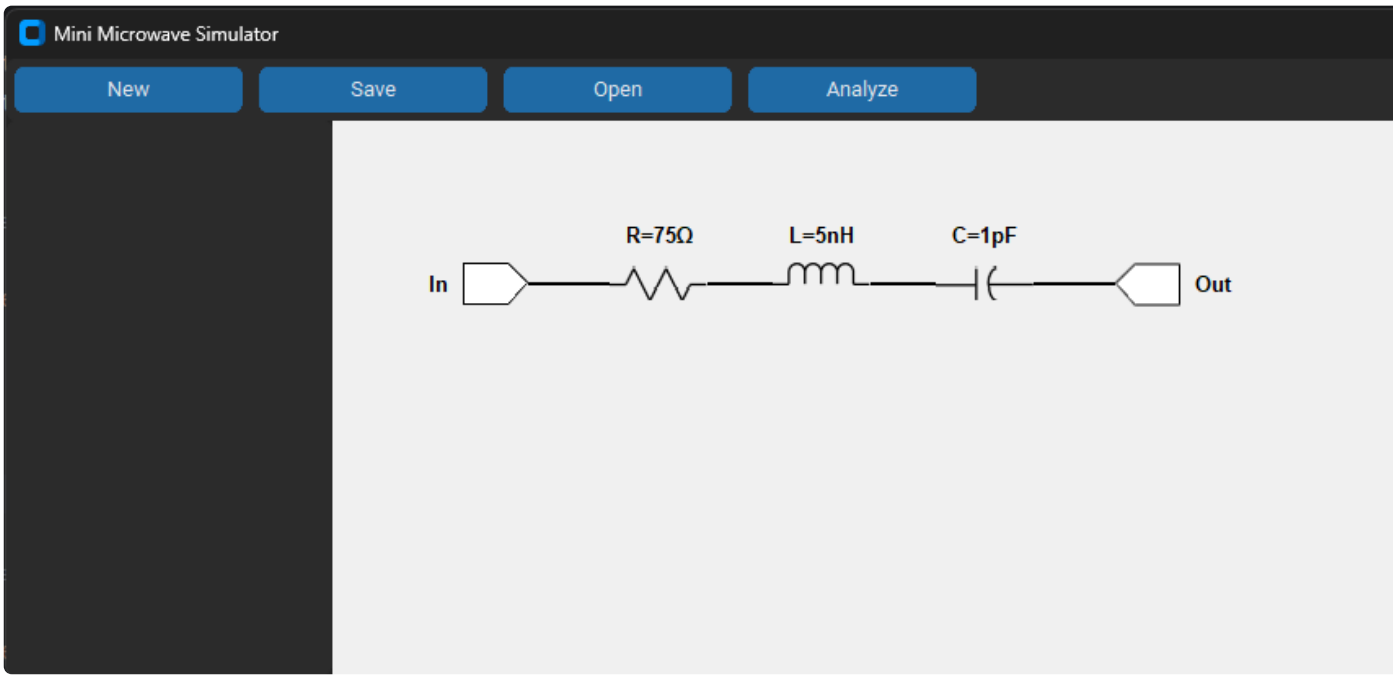
```
from Wire_Lib.wire_selector import WireSelector


class Wire:
    """Base class for wire classes"""
    def __init__(self, canvas, x1, y1, x2, y2):
        self.canvas = canvas
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

        self.fill_color = "black"
        self.border_width = 2
        self.line_direction = "horizontal"

        self.id = None
        self.is_selected = False
        self.selector = None
        self.width = 2
        self.bbox = None

        # Connections for wire list
        self.in_cnx = None  # Added
        self.out_cnx = None  # Added
        self.cnx = []  # Added

        self.sel1, self.sel2 = None, None
```

```
        . . .

    def create_wire_list_cnx(self, comp, end):  # Added new method - called
from Mouse Class
        if end == 'out':
            self.out_cnx = (comp, end)
        elif end == 'in1':
            self.in_cnx = (comp, end)
        if self.in_cnx and self.out_cnx:
            self.cnx = [self.out_cnx, self.in_cnx]
            self.canvas.wire_list.append(self.cnx)
```

UI_Lib/mouse.py

```
        . . .

    def select_connector(self, wire_obj, wire_end, x, y):  # Added method to
see if line end hits a gate connector
        for comp in self.canvas.comp_list:
            if not (isinstance(comp, StraightWire) or isinstance(comp,
SegmentWire) or
                        isinstance(self.selected_comp, ElbowWire)):
                conn = comp.check_connector_hit(x, y)
                if conn:
                    if wire_end == "begin":
                        wire_obj.x1, wire_obj.y1 = conn.x, conn.y
                    elif wire_end == "end":
                        wire_obj.x2, wire_obj.y2 = conn.x, conn.y
                    a_conn = Connection(conn, self.current_wire_obj, wire_end)
                    wire_obj.create_wire_list_cnx(comp, conn.name)  # Added
method call to creeate a wire connection (cnx)
                    comp.wire_list.append(a_conn)
                    self.canvas.redraw()
```

Mini Microwave Simulator


Mini Microwave Simulator
New
Save
Open
Analyze
R=75Ω
L=5nH
C=1pF
In
Out

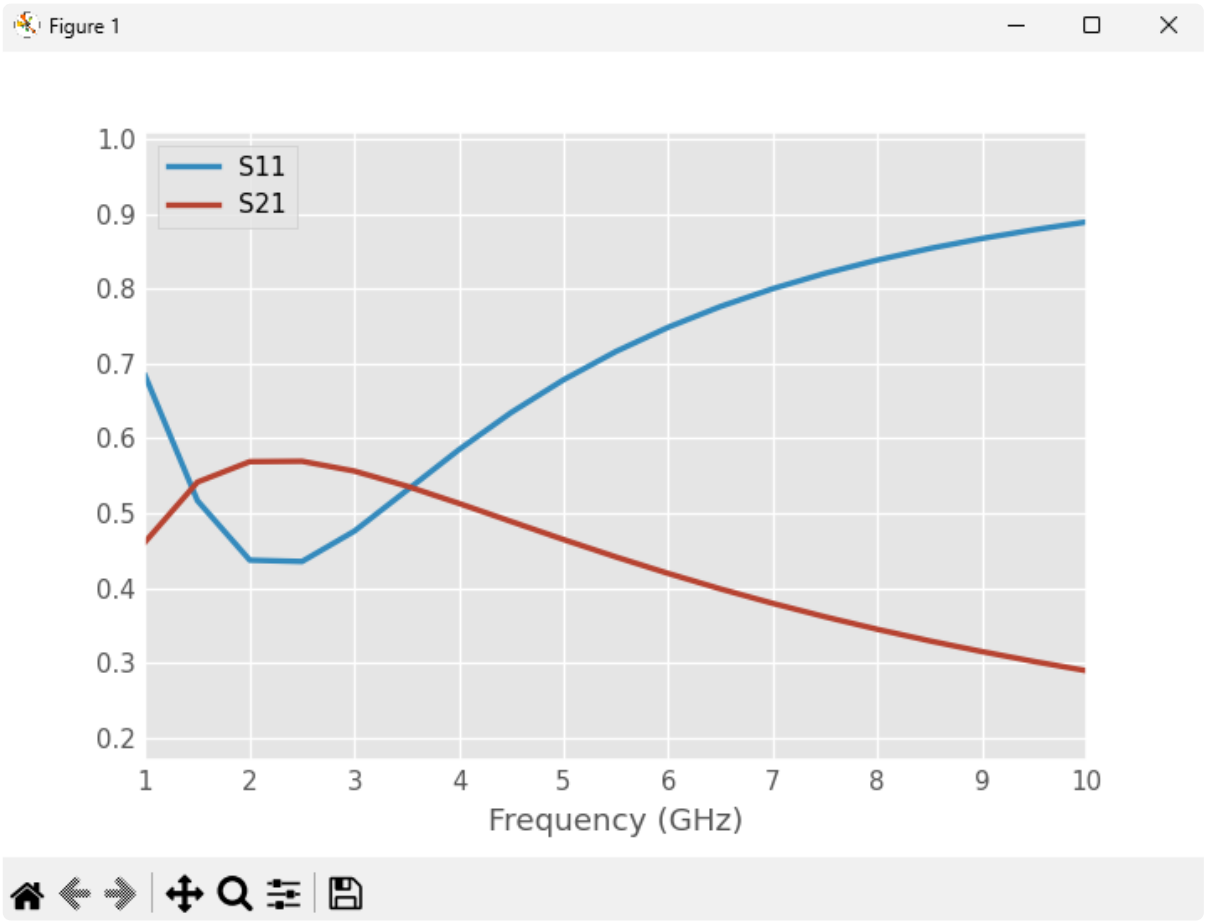


Figure 1
S11
S21
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
1
2
3
4
5
6
7
8
9
10
Frequency (GHz)

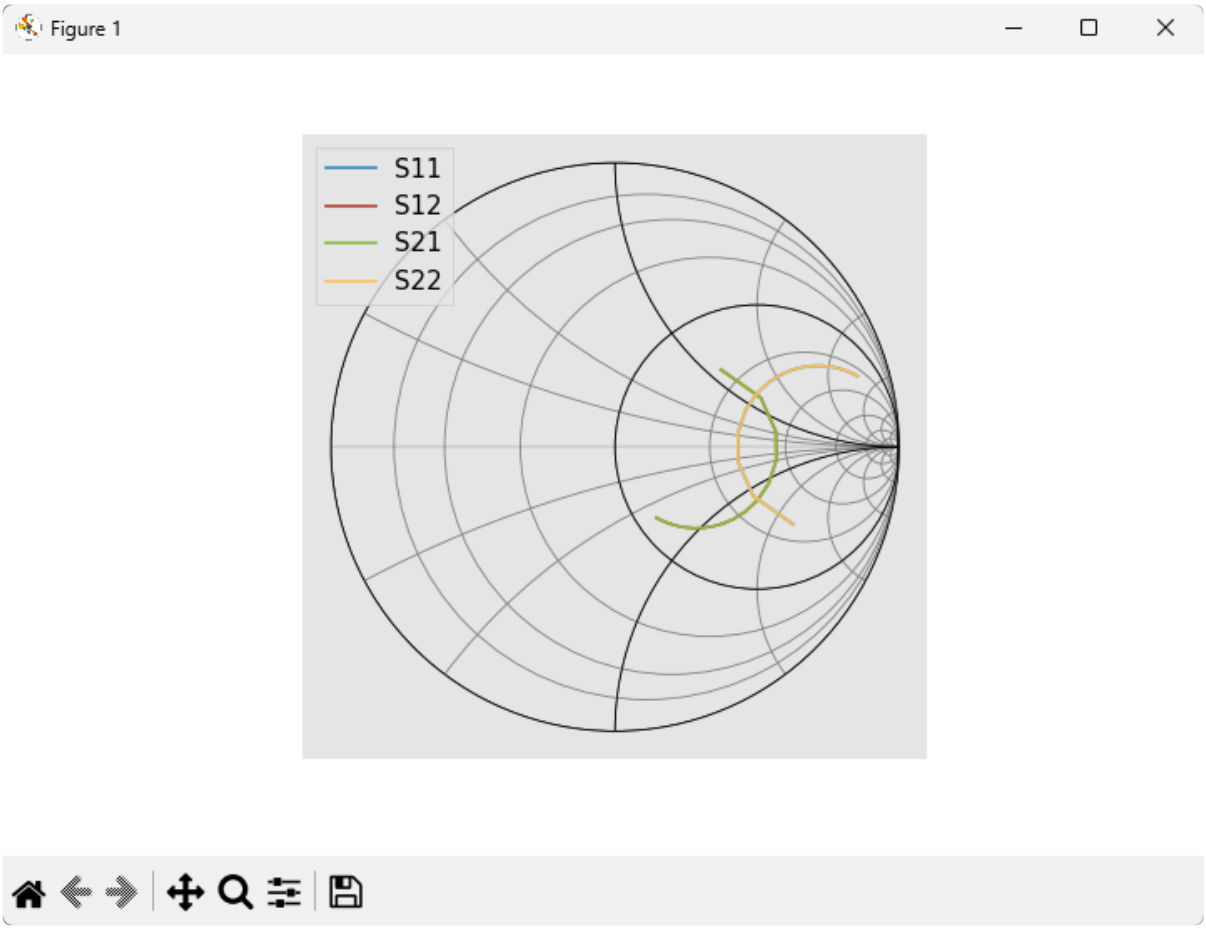


Figure 1
S11
S12
S21
S22

Console output

```
D:\EETools\MicrowaveSimulator\venv\Scripts\python.exe
D:\EETools\MicrowaveSimulator\Sandbox\mini_microwave_simulator.py
[[[0.68368911 0.46154922]
  [0.46154922 0.68368911]]

 [[0.51666132 0.54150201]
  [0.54150201 0.51666132]]

 [[0.43712731 0.56883028]
  [0.56883028 0.43712731]]

 [[0.43535197 0.56937463]
  [0.56937463 0.43535197]]

 [[0.47595789 0.55622445]
  [0.55622445 0.47595789]]

 [[0.53033858 0.5361869 ]
  [0.5361869  0.53033858]]

 [[0.58489041 0.51299248]
  [0.51299248 0.58489041]]

 [[0.63466307 0.48875466]
  [0.48875466 0.63466307]]

 [[0.67836989 0.46467808]
  [0.46467808 0.67836989]]

 [[0.71612899 0.4414337 ]
  [0.4414337  0.71612899]]

 [[0.7485435  0.41937221]
  [0.41937221 0.7485435 ]]

 [[0.77633296 0.39865129]
  [0.39865129 0.77633296]]
```

```
 [[0.80018956 0.37931342]
  [0.37931342 0.80018956]]

 [[0.82072802 0.36133393]
  [0.36133393 0.82072802]]

 [[0.83847445 0.34465089]
  [0.34465089 0.83847445]]

 [[0.85387096 0.32918346]
  [0.32918346 0.85387096]]

 [[0.86728573 0.31484312]
  [0.31484312 0.86728573]]

 [[0.87902421 0.30154034]
  [0.30154034 0.87902421]]

 [[0.88933967 0.28918848]
  [0.28918848 0.88933967]]]
```

Sandbox/mini_microwave_simulator.py

```
import customtkinter as ctk
from Comp_Lib import Inport, Outport, Resistor, Inductor, Capacitor
from Wire_Lib import StraightWire

import skrf as rf
import matplotlib.pyplot as plt
rf.stylely()

import json
from tkinter import filedialog as fd


class Encoder(json.JSONEncoder):
    def default(self, o):
        if hasattr(o, "reprJson"):
            return o.reprJson()
        else:
```

```
            return super().default(o)


class Decoder(json.JSONDecoder):
    def __init__(self):
        json.JSONDecoder.__init__(self, object_hook=Decoder.from_dict)

    @staticmethod
    def from_dict(_d):
        return _d


class Connection:
    def __init__(self, comp_conn, wire_name, wire_end):
        self.type = "connection"
        self.connector_obj = comp_conn   # in1, out
        self.wire_obj = wire_name        # wire string name
        self.wire_end = wire_end         # "begin" or "end"

    def reprJson(self):
        return dict(type=self.type, connector_obj=self.connector_obj,
wire_obj=self.wire_obj, wire_end=self.wire_end)


class App(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.geometry("1200x800x100x100")  # w, h, x, y
        self.title("Mini Microwave Simulator")

        self.canvas = ctk.CTkCanvas(self)
        self.left_frame = ctk.CTkFrame(self)
        self.top_frame = ctk.CTkFrame(self)

        self.top_frame.pack(side=ctk.TOP, fill=ctk.BOTH)
        self.left_frame.pack(side=ctk.LEFT, fill=ctk.BOTH)
        self.canvas.pack(side=ctk.LEFT, fill=ctk.BOTH, expand=True)

        # Add frame widgets here
        button = ctk.CTkButton(self.top_frame, text="New", command=self.new)
        button.pack(side=ctk.LEFT, padx=5, pady=5)

        button = ctk.CTkButton(self.top_frame, text="Save",
command=self.save_file)
        button.pack(side=ctk.LEFT, padx=5, pady=5)
```

```
        button = ctk.CTkButton(self.top_frame, text="Open",
command=self.open_file)
        button.pack(side=ctk.LEFT, padx=5, pady=5)

        button = ctk.CTkButton(self.top_frame, text="Analyze",
command=self.analyze)
        button.pack(side=ctk.LEFT, padx=5, pady=5)

        # Add component list here
        port1 = Inport(self.canvas, 100, 100)
        R1 = Resistor(self.canvas, 200, 100, 75)
        L1 = Inductor(self.canvas, 300, 100, 5)
        C1 = Capacitor(self.canvas, 400, 100, 1)
        port2 = Outport(self.canvas, 500, 100)
        wire1 = StraightWire(self.canvas, 120, 100, 170, 100)
        wire2 = StraightWire(self.canvas, 230, 100, 270, 100)
        wire3 = StraightWire(self.canvas, 330, 100, 370, 100)
        wire4 = StraightWire(self.canvas, 430, 100, 480, 100)
        self.comp_list = [port1, R1, L1, C1, port2, wire1, wire2, wire3, wire4]
        self.comp_dict = {'comp_list': self.comp_list}

        # Add wire connections here
        port1.wire_list.append(Connection('out', 'wire1', 'begin'))
        R1.wire_list.append(Connection('in1', 'wire1', 'end'))
        R1.wire_list.append(Connection('out', 'wire2', 'begin'))
        L1.wire_list.append(Connection('in1', 'wire2', 'end'))
        L1.wire_list.append(Connection('out', 'wire3', 'begin'))
        C1.wire_list.append(Connection('in1', 'wire3', 'end'))
        C1.wire_list.append(Connection('out', 'wire4', 'begin'))
        port2.wire_list.append(Connection('in1', 'wire4', 'end'))

        # Add Scikit-RF connections here
        conn1 = [('inport', 0), ('resistor', 0)]
        conn2 = [('resistor', 1), ('inductor', 0)]
        conn3 = [('inductor', 1), ('capacitor', 0)]
        conn4 = [('capacitor', 1), ('outport', 0)]
        self.conn_list = [conn1, conn2, conn3, conn4]
        self.conn_dict = {'conn_list': self.conn_list}

        self.circuit = [self.comp_dict, self.conn_dict]

    def new(self):
        self.comp_list.clear()
        self.conn_list.clear()
        self.canvas.delete('all')
```

```
    def save_file(self):
        filetypes = (('json files', '*.json'), ('All files', '*.*'))
        f = fd.asksaveasfilename(filetypes=filetypes, initialdir="./")
        with open(f, 'w') as file:
            file.write(json.dumps(self.circuit, cls=Encoder, indent=4))

    def open_file(self):
        try:
            filetypes = (('json files', '*.json'), ('All files', '*.*'))
            f = fd.askopenfilename(filetypes=filetypes, initialdir="./")
            with open(f) as file:
                d = json.load(file)
                self.convert_json_data(d)
        except FileNotFoundError:
            with open('untitled.canvas', 'w') as _file:
                pass

    def convert_json_data(self, data):
        """Convert json data to a circuit object"""
        # Get circuit list from json data
        json_comp_list = data[0]['comp_list']
        for json_comp in json_comp_list:
            if json_comp['type'] == 'resistor':
                res = Resistor(self.canvas, int(json_comp['x1']),
int(json_comp['y1']), int(json_comp['resistance']))
                res.wire_list = json_comp['wire_list']
                self.comp_list.append(res)
            elif json_comp['type'] == 'inductor':
                ind = Inductor(self.canvas, int(json_comp['x1']),
int(json_comp['y1']), int(json_comp['inductance']))
                ind.wire_list = json_comp['wire_list']
                self.comp_list.append(ind)
            elif json_comp['type'] == 'capacitor':
                cap = Capacitor(self.canvas, int(json_comp['x1']),
int(json_comp['y1']), int(json_comp['capacitance']))
                cap.wire_list = json_comp['wire_list']
                self.comp_list.append(cap)
            elif json_comp['type'] == 'inport':
                port1 = Inport(self.canvas, int(json_comp['x1']),
int(json_comp['y1']))
                port1.wire_list = json_comp['wire_list']
                self.comp_list.append(port1)
            elif json_comp['type'] == 'outport':
                port2 = Outport(self.canvas, int(json_comp['x1']),
int(json_comp['y1']))
                port2.wire_list = json_comp['wire_list']
```

```
                self.comp_list.append(port2)
            elif json_comp['type'] == 'straight':
                wire = StraightWire(self.canvas, int(json_comp['x1']),
int(json_comp['y1']), int(json_comp['x2']),
                                    int(json_comp['y2']))
                self.comp_list.append(wire)

        # Get connection list from json data
        self.conn_list = data[1]['conn_list']

    def analyze(self):
        ctx = []
        res, ind, cap, inport, outport = None, None, None, None, None
        freq = rf.Frequency(start=1.0, stop=10.0, unit='GHz', npoints=19)

        # Convert graphical components to microwave components
        for comp in self.comp_list:
            if isinstance(comp, Resistor):
                res = rf.Circuit.SeriesImpedance(frequency=freq, name='res',
z0=50, Z=comp.resistance)
            elif isinstance(comp, Inductor):
                ind = rf.Circuit.SeriesImpedance(frequency=freq, name='ind',
z0=50,
                                                 Z=1j * freq.w *
comp.inductance * 1e-9)
            elif isinstance(comp, Capacitor):
                cap = rf.Circuit.SeriesImpedance(frequency=freq, name='cap',
z0=50,
                                                 Z=1 / (1j * freq.w *
comp.capacitance * 1e-12))
            elif isinstance(comp, Inport):
                inport = rf.Circuit.Port(freq, name='inport', z0=50)
            elif isinstance(comp, Outport):
                outport = rf.Circuit.Port(freq, name='outport', z0=50)

        comp_dict = {
            'resistor': res,
            'inductor': ind,
            'capacitor': cap,
            'inport': inport,
            'outport': outport,
            'wire': None
        }

        for conn in self.conn_list:
            ctx_conn = [(comp_dict[conn[0][0]], conn[0][1]), (comp_dict[conn[1]
```

```
[0]], conn[1][1])]
            ctx.append(ctx_conn)

        ckt = rf.Circuit(ctx)
        ntw = ckt.network

        print(ntw.s_mag)  # Print S-parameters to console

        ntw.plot_s_mag(m=0, n=0, lw=2, logx=False)
        ntw.plot_s_mag(m=1, n=0, lw=2, logx=False)
        plt.show()  # Display S-parameter plot

        ntw.plot_s_smith()
        plt.show()


if __name__ == "__main__":
    app = App()
    app.mainloop()
```

Sandbox/`RLC.json`

```
[
    {
        "comp_list": [
            {
                "type": "inport",
                "x1": 100,
                "y1": 100,
                "angle": 0,
                "wire_list": [
                    {
                        "type": "connection",
                        "connector_obj": "out",
                        "wire_obj": "wire1",
                        "wire_end": "begin"
                    }
                ]
            },
            {
                "type": "resistor",
                "x1": 200,
                "y1": 100,
```

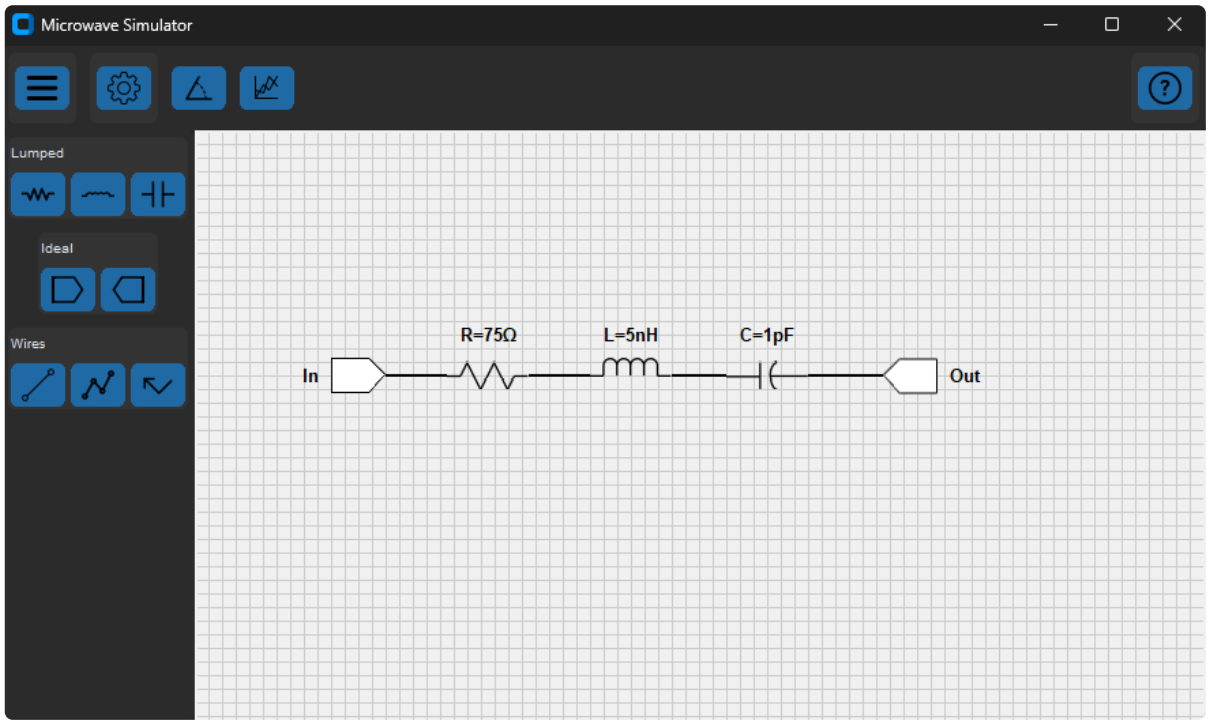
```
            "angle": 0,
            "resistance": 75,
            "wire_list": [
                {
                    "type": "connection",
                    "connector_obj": "in1",
                    "wire_obj": "wire1",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "connector_obj": "out",
                    "wire_obj": "wire2",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "inductor",
            "x1": 300,
            "y1": 100,
            "angle": 0,
            "inductance": 5,
            "wire_list": [
                {
                    "type": "connection",
                    "connector_obj": "in1",
                    "wire_obj": "wire2",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "connector_obj": "out",
                    "wire_obj": "wire3",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "capacitor",
            "x1": 400,
            "y1": 100,
            "angle": 0,
            "capacitance": 1,
            "wire_list": [
                {
```

```
                    "type": "connection",
                    "connector_obj": "in1",
                    "wire_obj": "wire3",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "connector_obj": "out",
                    "wire_obj": "wire4",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "outport",
            "x1": 500,
            "y1": 100,
            "angle": 0,
            "wire_list": [
                {
                    "type": "connection",
                    "connector_obj": "in1",
                    "wire_obj": "wire4",
                    "wire_end": "end"
                }
            ]
        },
        {
            "type": "straight",
            "x1": 120,
            "y1": 100,
            "x2": 170,
            "y2": 100
        },
        {
            "type": "straight",
            "x1": 230,
            "y1": 100,
            "x2": 270,
            "y2": 100
        },
        {
            "type": "straight",
            "x1": 330,
            "y1": 100,
            "x2": 370,
```

```
                "y2": 100
            },
            {
                "type": "straight",
                "x1": 430,
                "y1": 100,
                "x2": 480,
                "y2": 100
            }
        ]
    },
    {
        "conn_list": [
            [
                [
                    "inport",
                    0
                ],
                [
                    "resistor",
                    0
                ]
            ],
            [
                [
                    "resistor",
                    1
                ],
                [
                    "inductor",
                    0
                ]
            ],
            [
                [
                    "inductor",
                    1
                ],
                [
                    "capacitor",
                    0
                ]
            ],
            [
                [
                    "capacitor",
```

```
                    1
                ],
                [
                    "outport",
                    0
                ]
            ]
        ]
    }
]
```

RLC File Save & Load



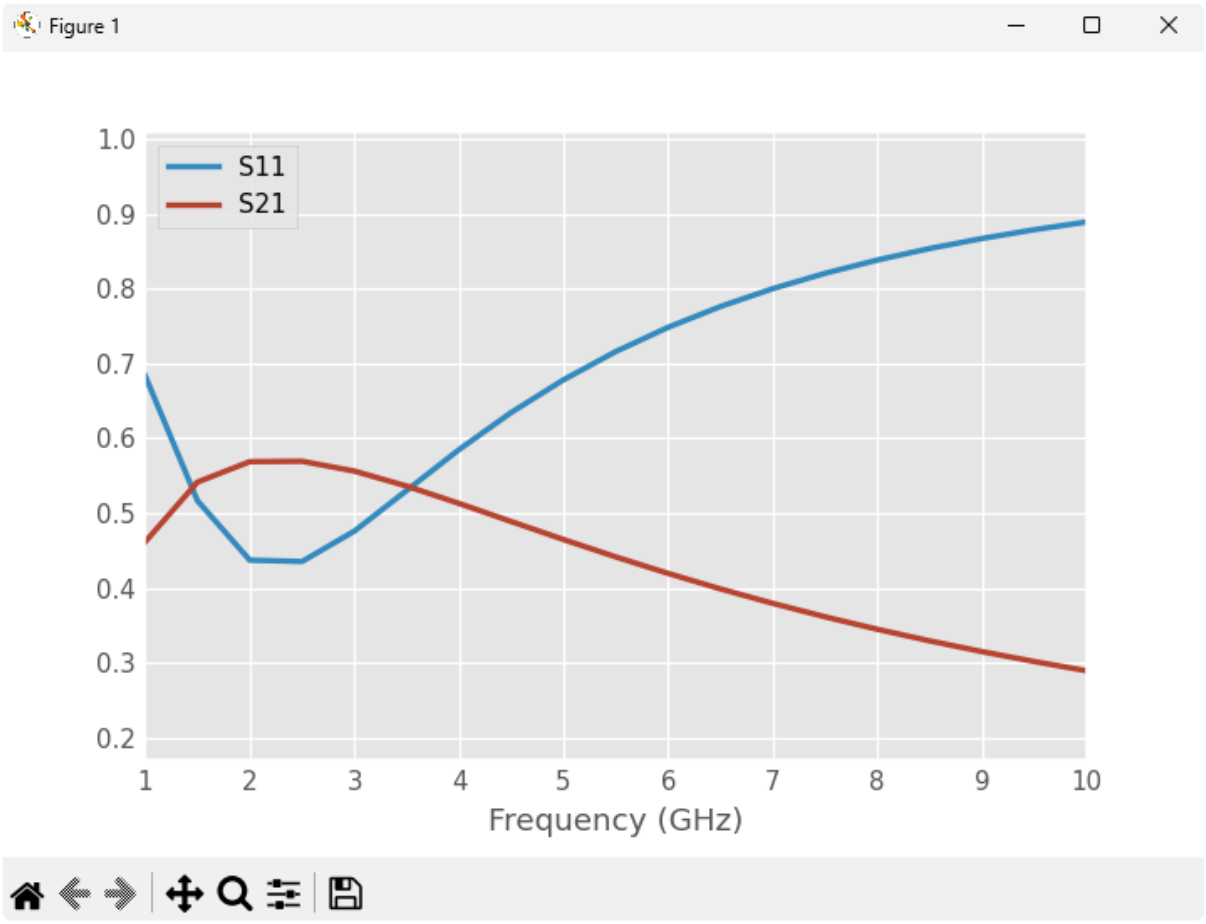


Figure 1
S11
S21
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
1
2
3
4
5
6
7
8
9
10
Frequency (GHz)

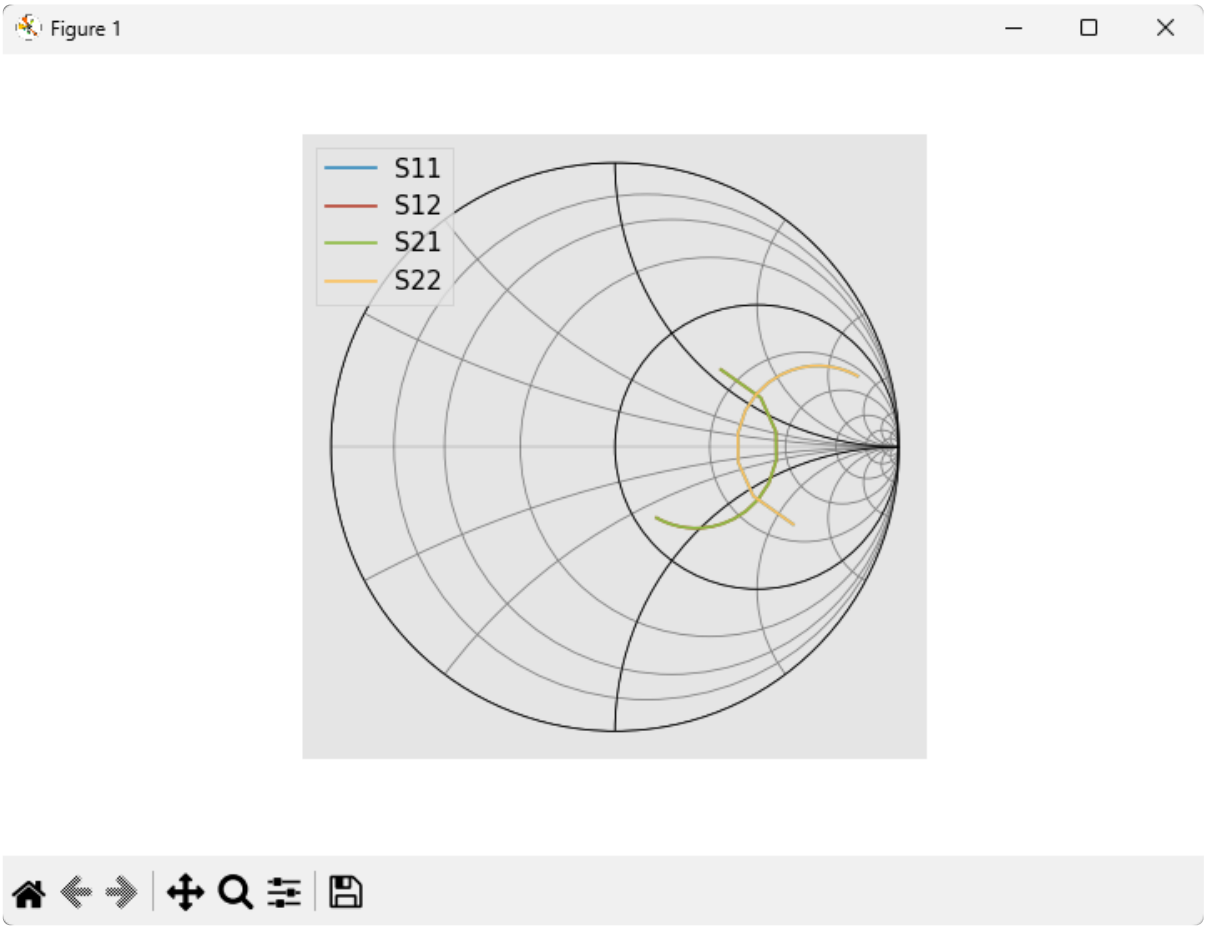


Figure 1
S11
S12
S21
S22

Console output

```
D:\EETools\MicrowaveSimulator\venv\Scripts\python.exe
D:\EETools\MicrowaveSimulator\Sandbox\mini_microwave_simulator.py
[[[0.68368911 0.46154922]
  [0.46154922 0.68368911]]

 [[0.51666132 0.54150201]
  [0.54150201 0.51666132]]

 [[0.43712731 0.56883028]
  [0.56883028 0.43712731]]

 [[0.43535197 0.56937463]
  [0.56937463 0.43535197]]

 [[0.47595789 0.55622445]
  [0.55622445 0.47595789]]

 [[0.53033858 0.5361869 ]
  [0.5361869  0.53033858]]

 [[0.58489041 0.51299248]
  [0.51299248 0.58489041]]

 [[0.63466307 0.48875466]
  [0.48875466 0.63466307]]

 [[0.67836989 0.46467808]
  [0.46467808 0.67836989]]

 [[0.71612899 0.4414337 ]
  [0.4414337  0.71612899]]

 [[0.7485435  0.41937221]
  [0.41937221 0.7485435 ]]

 [[0.77633296 0.39865129]
  [0.39865129 0.77633296]]
```

```
 [[0.80018956 0.37931342]
  [0.37931342 0.80018956]]

 [[0.82072802 0.36133393]
  [0.36133393 0.82072802]]

 [[0.83847445 0.34465089]
  [0.34465089 0.83847445]]

 [[0.85387096 0.32918346]
  [0.32918346 0.85387096]]

 [[0.86728573 0.31484312]
  [0.31484312 0.86728573]]

 [[0.87902421 0.30154034]
  [0.30154034 0.87902421]]

 [[0.88933967 0.28918848]
  [0.28918848 0.88933967]]]
```

UI_Lib/wire_button_frame.py

```
import customtkinter as ctk
from pathlib import Path
from PIL import Image

from Wire_Lib import StraightWire, SegmentWire, ElbowWire


class WireButtonFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.wire = None
        self.wire_count = 0  # Added variable to keep track of wire number
count

        # Add frame widgets here
        frame_name_label = ctk.CTkLabel(self, text="Wires", font=("Helvetica",
```

```
10), height=20)
        frame_name_label.grid(row=0, column=0, columnspan=3, sticky=ctk.W,
padx=2, pady=2)

        straight_wire_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/straight_line.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/straight_line.png"),
                                size=(24, 24))

        straight_wire_button = ctk.CTkButton(self, text="",
image=straight_wire_image, width=30,
                                        command=self.create_straight_wire)
        straight_wire_button.grid(row=1, column=0, sticky=ctk.W, padx=2,
pady=2)

        segment_wire_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/segment_line.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/segment_line.png"),
                                size=(24, 24))

        segment_wire_button = ctk.CTkButton(self, text="",
image=segment_wire_image, width=30,
                                        command=self.create_segment_wire)
        segment_wire_button.grid(row=1, column=1, sticky=ctk.W, padx=2, pady=2)

        elbow_wire_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/elbow_line.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/elbow_line.png"),
                                size=(24, 24))

        elbow_wire_button = ctk.CTkButton(self, text="",
image=elbow_wire_image, width=30,
                                        command=self.create_elbow_wire)
        elbow_wire_button.grid(row=1, column=2, sticky=ctk.W, padx=2, pady=2)

    # Shape button handlers
    def create_straight_wire(self):
```

```
        wire = StraightWire(self.canvas, 0, 0, 0, 0)
        self.create_wire(wire)

    def create_segment_wire(self):
        wire = SegmentWire(self.canvas, 0, 0, 0, 0)
        self.create_wire(wire)

    def create_elbow_wire(self):
        wire = ElbowWire(self.canvas, 0, 0, 0, 0)
        self.create_wire(wire)

    def create_wire(self, wire):
        self.assign_wire_name(wire)  # Add method call to assign a wire name
        self.canvas.mouse.current_wire_obj = wire
        self.canvas.show_connectors()
        self.canvas.comp_list.append(wire)
        self.canvas.mouse.draw_wire_mouse_events()

    def assign_wire_name(self, wire):  # Added method to create and assign a
wire name
        self.wire_count += 1
        wire_name = 'wire' + str(self.wire_count)
        wire.name = wire_name
        self.canvas.wire_dict[wire_name] = wire
```

UI_Lib/file_menu_frame.py

```
import customtkinter as ctk
from tkinter import filedialog as fd
from pathlib import Path
import json
from PIL import Image

from Comp_Lib import Resistor, Inductor, Capacitor, Connection
from Comp_Lib import Inport, Outport
from Wire_Lib import StraightWire, SegmentWire, ElbowWire


class Encoder(json.JSONEncoder):
    def default(self, o):
        if hasattr(o, "reprJson"):
            return o.reprJson()
        else:
```

```
            return super().default(o)


class Decoder(json.JSONDecoder):
    def __init__(self):
        json.JSONDecoder.__init__(self, object_hook=Decoder.from_dict)

    @staticmethod
    def from_dict(_d):
        return _d


class FileMenuFrame(ctk.CTkFrame):
    def __init__(self, window, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.obj_type_dict = {'resistor': Resistor,
                              'inductor': Inductor,
                              'capacitor': Capacitor,
                              'inport': Inport,
                              'outport': Outport,
                              'straight': StraightWire,
                              'segment': SegmentWire,
                              'elbow': ElbowWire}

        self.menu_on = False

        self.menu_frame = ctk.CTkFrame(window, height=100, bg_color="white")

        new_btn = ctk.CTkButton(self.menu_frame, text="New", width=150,
command=self.new_diagram)
        new_btn.pack(pady=5)

        open_btn = ctk.CTkButton(self.menu_frame, text="Open", width=150,
command=self.load_diagram)
        open_btn.pack(pady=5)

        save_btn = ctk.CTkButton(self.menu_frame, text="Save", width=150,
command=self.save_diagram)
        save_btn.pack(pady=5)

        exit_btn = ctk.CTkButton(self.menu_frame, text="Exit", width=150,
command=window.destroy)
        exit_btn.pack(pady=5)
```

```
        my_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/hamburger_menu.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/hamburger_menu.png"),
                                size=(24, 24))

        button = ctk.CTkButton(self, text="", image=my_image, width=30,
command=self.show_menu)
        button.pack(side=ctk.LEFT, padx=5, pady=10)

    def new_diagram(self):
        self.canvas.delete("all")
        self.canvas.comp_list.clear()
        self.canvas.wire_list.clear()

    def load_diagram(self):
        try:
            filetypes = (('json files', '*.json'), ('All files', '*.*'))
            f = fd.askopenfilename(filetypes=filetypes, initialdir="./")
            with open(f) as file:
                d = json.load(file)
                self.convert_json_data(d)
        except FileNotFoundError:
            with open('untitled.canvas', 'w') as _file:
                pass

    def convert_json_data(self, data):
        """Convert json data to a circuit object"""
        # Get circuit list from json data
        json_comp_list = data[0]['comp_list']
        for json_comp in json_comp_list:
            if json_comp['type'] == 'resistor':
                res = Resistor(self.canvas, int(json_comp['x1']),
int(json_comp['y1']), int(json_comp['resistance']))
                conn_dict = json_comp['wire_list'][0]
                res.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                conn_dict = json_comp['wire_list'][1]
                res.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                self.canvas.comp_list.append(res)
            elif json_comp['type'] == 'inductor':
```

```
                ind = Inductor(self.canvas, int(json_comp['x1']),
int(json_comp['y1']), int(json_comp['inductance']))
                conn_dict = json_comp['wire_list'][0]
                ind.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                conn_dict = json_comp['wire_list'][1]
                ind.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                self.canvas.comp_list.append(ind)
            elif json_comp['type'] == 'capacitor':
                cap = Capacitor(self.canvas, int(json_comp['x1']),
int(json_comp['y1']), int(json_comp['capacitance']))
                conn_dict = json_comp['wire_list'][0]
                cap.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                conn_dict = json_comp['wire_list'][1]
                cap.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                self.canvas.comp_list.append(cap)
            elif json_comp['type'] == 'inport':
                port1 = Inport(self.canvas, int(json_comp['x1']),
int(json_comp['y1']))
                conn_dict = json_comp['wire_list'][0]
                port1.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                self.canvas.comp_list.append(port1)
            elif json_comp['type'] == 'outport':
                port2 = Outport(self.canvas, int(json_comp['x1']),
int(json_comp['y1']))
                conn_dict = json_comp['wire_list'][0]
                port2.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                self.canvas.comp_list.append(port2)
            elif json_comp['type'] == 'straight':
                wire = StraightWire(self.canvas, int(json_comp['x1']),
int(json_comp['y1']), int(json_comp['x2']),
                                    int(json_comp['y2']))
                wire.name = json_comp['name']
                self.canvas.comp_list.append(wire)
                self.canvas.wire_dict[wire.name] = wire
```

```
        # Get connection list from json data
        self.canvas.conn_list = data[1]['conn_list']

    def save_diagram(self):
        comp_dict = {'comp_list': self.canvas.comp_list}
        conn_dict = {'conn_list': self.canvas.conn_list}
        circuit = [comp_dict, conn_dict]

        filetypes = (('json files', '*.json'), ('All files', '*.*'))
        f = fd.asksaveasfilename(filetypes=filetypes, initialdir="./")
        with open(f, 'w') as file:
            file.write(json.dumps(circuit, cls=Encoder, indent=4))

    def show_menu(self):
        if not self.menu_on:
            self.menu_frame.place(x=5, y=60)
            self.menu_frame.tkraise()
            self.menu_on = True
        else:
            self.menu_frame.place_forget()
            self.menu_on = False
```

Circuits/RLC.json

```
[
    {
        "comp_list": [
            {
                "type": "inport",
                "x1": 120,
                "y1": 180,
                "angle": 0,
                "wire_list": [
                    {
                        "type": "connection",
                        "comp_conn": "out",
                        "wire_name": "wire1",
                        "wire_end": "begin"
                    }
                ]
            },
            {
```

```
            "type": "resistor",
            "x1": 215,
            "y1": 180,
            "angle": 0,
            "resistance": 75,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire1",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire2",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "inductor",
            "x1": 320,
            "y1": 180,
            "angle": 0,
            "inductance": 5,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire2",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire3",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "capacitor",
            "x1": 420,
            "y1": 180,
            "angle": 0,
```

```
            "capacitance": 1,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire3",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire4",
                    "wire_end": "begin"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire4",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire5",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "outport",
            "x1": 525,
            "y1": 180,
            "angle": 0,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire5",
                    "wire_end": "end"
                }
            ]
        },
        {
            "type": "straight",
            "x1": 140.0,
            "y1": 180.0,
```

```
                "x2": 185.0,
                "y2": 180.0,
                "name": "wire1"
            },
            {
                "type": "straight",
                "x1": 245.0,
                "y1": 180.0,
                "x2": 290.0,
                "y2": 180.0,
                "name": "wire2"
            },
            {
                "type": "straight",
                "x1": 350.0,
                "y1": 180.0,
                "x2": 390.0,
                "y2": 180.0,
                "name": "wire3"
            },
            {
                "type": "straight",
                "x1": 450.0,
                "y1": 180.0,
                "x2": 450.0,
                "y2": 180.0,
                "name": "wire4"
            },
            {
                "type": "straight",
                "x1": 450.0,
                "y1": 180.0,
                "x2": 505.0,
                "y2": 180.0,
                "name": "wire5"
            }
        ]
    },
    {
        "conn_list": [
            [
                [
                    "inport",
                    0
                ],
                [
```

```
                    "resistor",
                    0
                ]
            ],
            [
                [
                    "resistor",
                    1
                ],
                [
                    "inductor",
                    0
                ]
            ],
            [
                [
                    "inductor",
                    1
                ],
                [
                    "capacitor",
                    0
                ]
            ],
            [
                [
                    "capacitor",
                    1
                ],
                [
                    "outport",
                    0
                ]
            ]
        ]
    }
]
```

Summary

Microwave circuit simulation using the Scikit-RF library provides a good basis for RF and microwave simulators. This concludes the development of our third advanced electrical engineering project. The final project will be an Analog Circuit Simulator using many of the

techniques we development for the Microwave Circuit Simulator. If you made it this far, you are ready to complete the last tool in the EE Tools software. See you in the next chapter.