Design and Implementation of an Object-Oriented AC Circuit Simulator in C++

Joshua Holmes
The University of Manchester
(Dated: May 15, 2023)

This paper describes the design and implementation of an AC circuit simulator using object-oriented programming techniques in C++. The simulator allows for the creation, modification, and analysis of complex AC circuits, featuring a variety of components including resistors, capacitors, inductors, diodes, and transistors. The software is capable of handling both series and parallel circuit configurations, as well as nested circuits, offering a flexible and intuitive approach for circuit assembly. User input validation is rigorously enforced to ensure robustness, while an organized division of classes into separate header and implementation files enhances maintainability. The developed software, with its comprehensive features, serves as an effective educational tool for understanding the fundamentals of AC circuits and their behavior.

I. INTRODUCTION:

Analogue circuits are circuits that make use of continuous signals to transmit information as opposed to discrete signals in digital circuits. Circuits making use of *alternating-currents* (AC) are in widespread use throughout the world for both household and industrial applications. As a result, an understanding of AC circuits is important for maximising the efficiency of such products [1, 2].

The primary elements of AC circuit theory encompass understanding the behavior of resistors, capacitors, and inductors under alternating current conditions [3]. Resistors are characterized by a linear relationship between voltage and current, represented by their resistance (R). Conversely, capacitors, characterized by capacitance (C), and inductors, characterized by inductance (L), store and release energy back into the circuit, leading to a phase difference between voltage and current.

Impedance is a crucial concept in AC circuits, extending the notion of resistance to include capacitors and inductors. For resistors, impedance is equivalent to resistance and is a real value. However, for capacitors and inductors, impedance is a complex value and is frequency-dependent. Specifically, capacitive impedance decreases as frequency increases, while inductive impedance increases with frequency. The equation for impedance is given by

$$Z = R + Xj, \tag{1}$$

where Z is the impedance, R is the resistance, X is the reactance, and j is the imaginary unit, $\sqrt{-1}$ [4]. The impedance for a resistor has no imaginary component and is simply given by

$$Z_R = R. \tag{2}$$

In contrast to this, the impedance of a capacitor is purely imaginary, and is given by:

$$Z_C = \frac{1}{j\omega C}, \tag{3}$$

while the impedance of an inductor is given by:

$$Z_I = j\omega L, \tag{4}$$

where ω is the angular frequency of the respective component [5].

Following on from this, in order to calculate the impedance of a number of components in a series combination, the following equation is used:

$$Z_{eq} = Z_1 + Z_2 + ... + Z_n, \tag{5}$$

where n is the total number of components added in series. Similarly, the impedance of an arbitrary number of components in parallel can be calculated using

$$\frac{1}{Z_{eq}} = \frac{1}{Z_1} + \frac{1}{Z_2} + ... + \frac{1}{Z_n}, \tag{6}$$

[4] where, again, n is the number of components in parallel. Forward based diodes, which are considered in this project, have an idealised impedance of 0, but this is not realistic, most real diodes simply have very small resistances, so for our purposes, we will consider diodes to have a resistance of 1Ω.

Similarly, the transistor has an impedance calculated through it's gain, the current, and voltage drop [4], as these things aren't considered within this project, it's impedance is simply modelled by a user-inputted effective resistance.

$$Z_T = R_{eff} \tag{7}$$

II. CODE DESIGN AND IMPLEMENTATION

The design and implementation of our AC circuit simulator leverages the power of object-oriented programming (OOP) to create an interactive and versatile tool. The simulator is built around several key classes that represent various components of an AC circuit, encapsulating their properties and behaviors.

The **Component** abstract base class defines common attributes and methods for all components. This

class implements the interface for calculating impedance, phase difference, and magnitude. Each specific component (Resistor, Capacitor, Inductor, Diode, Transistor) is implemented as a derived class that overrides these methods as required.

The Component Library in the AC Circuit Simulator serves as a repository for predefined components that can be reused across multiple circuits. This not only enhances the modularity of the code but also allows for efficient memory usage since the same component doesn't need to be instantiated multiple times.

The Component Library is implemented using a std::map data structure where the key is the name of the component and the value is a **std::shared_ptr** pointing to the Component object. The use of **std::shared_ptr** ensures that the Component objects are deallocated when they're no longer in use, preventing memory leaks which are a common problem within C++ programs.

The implementation of nested circuits was achieved by treating a circuit itself as a component. This is facilitated by the **NestedCircuit** class, which inherits from both the Component and Circuit classes. This design allows a **NestedCircuit** object to behave both as a circuit (in terms of adding components to it) and as a component (in terms of being added to another circuit).

When a **NestedCircuit** object is added to a Circuit object, it is treated as a single component, and its impedance is calculated based on the impedances of the components it contains. This is done by overriding the **get_impedance** method in the **NestedCircuit** class to calculate and return the total impedance of the nested circuit.

The user interacts with the simulation through the use of a number of menus defined in the **main** function of the code. This uses recursivity to present the user with an arbitrary number of options for combining components in series and parallel. From that, the options to output the impedance of the circuit or show a depiction of the circuit are then present.

Input validation was achieved using standard library functions and loops. For example, the **get_menu_option** function, demonstrated in Listing 1, was designed to ensure that only integer inputs within a specified range are accepted. If the user enters an invalid input, the function continues to prompt the user until a valid input is received. This is done by taking the input as a string, checking for characters that aren't allowed, and then if the input has no non-integer elements, **return std::stoi(input)** converts it back into an **int** variable and passes it to the menu. Whether or not the selected input is within the required range is then handled within the respective menu implementation as there are multiple menus used, with different lengths.

```
int get_menu_option() {
    while (true) {
        std::cout << "Enter your choice: ";
        std::string input;
        std::cin >> input;

        if (std::all_of(input.begin(),
                input.end(), ::isdigit)) {return std::
                    stoi(input);
        }
        else {
            std::cout << "Invalid input. Please enter
                an integer in the specified range.\n";
        }
    }
}
```

Listing 1. This snippet demonstrates the function used to validate the inputs for menu options.

The **get_positive_number** function was designed to verify that user-entered values for component properties (resistance, capacitance, etc.) are valid. It checks that the input is a number and that it falls within a reasonable range for the particular property. If the input is invalid, the function will continue prompting the user until a valid value is entered. The purpose of this is that negative values for these inputs are unphysical and an impedance of 0 can lead to division by zero errors due to Eqn. (6).

Once the user has constructed a circuit to their specifications, they can then choose to display the impedances of both the circuit as a whole, and each individual component. This is achieved through the implementation of the Eqns. (2-7) and gives outputs both of the magnitude and phase difference of the impedance for the components, but also outputs the total impedance for the circuit as a complex number in the style of Eqn. (1). This is done through the use of the **Display_Complex** function.

The user can also choose to display a simple text-based representation of their circuit. This is done through the **CircuitVisualiser** class. This creates a string representation of the circuit, adding new components as they are connected in series or parallel, and finally displaying the entire circuit when the **display()** method is called. This is displyed in Listing 2.

```
void CircuitVisualiser::add_component_serial(std::
    shared_ptr<Component> component) {
    if (circuit_representation.empty()) {
        circuit_representation += " ";
    }
    else {
        circuit_representation += " --- ";
    }
    circuit_representation += component->representation
        ();
}

void CircuitVisualiser::add_component_parallel(std::
    shared_ptr<Component> component) {
    std::string parallel_bar = "|";
    std::string new_representation;
    std::istringstream iss(circuit_representation);
    std::string line;

    // Add parallel bars to the existing representation
    while (std::getline(iss, line)) {
        new_representation += parallel_bar + line +
            parallel_bar + "\n";
    }

    // Add the new component in parallel
    new_representation += parallel_bar + component->
        representation() + parallel_bar;

    circuit_representation = new_representation;
}
void CircuitVisualiser::display() const {
    std::cout << circuit_representation << std::endl;
```

```
}
```

Listing 2. This snippet demonstrates the function used to display the circuit once the user has finished constructing it.

III. RESULTS

An example final display of impedances is shown in Listing 3. The impedances of individual components are displayed in the order they were added to the circuit. The total calculated impedance matches with theoretical predictions. based on Section I.

```
The components are listed in the order you created them
    :
--------------------------------------
Impedance of component 1:
   Impedance: 600 Ohms
   Phase Difference: 0 Radians
--------------------------------------
Impedance of component 2:
   Impedance: 0.198944 Ohms
   Phase Difference: -1.5708 Radians
--------------------------------------
Impedance of component 3:
   Impedance: 150.796 Ohms
   Phase Difference: 1.5708 Radians
--------------------------------------
Impedance of component 4:
   Impedance: 600 Ohms
   Phase Difference: 0 Radians
--------------------------------------
--------------------------------------
Total circuit complex impedance: (60.518 + 120.387j)
    Ohms
   Impedance magnitude: 134.742 Ohms
   Phase Difference: 1.10499 Radians
--------------------------------------
--------------------------------------
```

Listing 3. This is an example output of impedances. In this instance, component 1 was an 600Ω resistor, component 2 was a 0.01 Farad, 80Hz capacitor added in series, component 3 was a 0.3 Henry, 80Hz inductor added in parallel, and component 4 was another 600Ω resistor added in parallel.

An example of a displayed circuit is given in Listing 4. It displays a fairly simple circuit with a number of different components connected in a combination of series and parallel connections. The series connections are denoted by a horizontal dashed line "—", and the parallel connections are denoted by being on a different vertical layer, and having two vertical bars at either side: "|...|". The individual components are denoted as follows:

- Resistor = [R]
- Capacitor = [C]
- Inductor = [L]
- Diode = [D]
- Transistor = [T]
- Nested Circuit = [N]

```
|| [R] --- [T]||
||[C]| --- [L] --- [D]|
|[C]|
```

Listing 4. This is an example output of the Circuit display feature. It shows a resistor and transistor connected in series, they are in parallel to a series-connected capacitor, inductor, and diode, and then they too are connected in parallel to a capacitor on it's own.

The internal structure of nested circuits cannot be displayed using this program.

IV. CONCLUSION AND DISCUSSION

In summary, this project has resulted in the creation of a simulator for AC circuits which gives accurate answers for the impedance of circuits with an arbitrary number of components and configurations. It can also display the circuits in a simple but clear way. This has been accomplished using code which has been constructed with careful memory allocation and input handling.

The primary limitations of this project are the fact that it only deals with ideal components, and specifically uses very simplified versions of the transistor and diode. Secondly, the ability to display the internal structure of nested circuits would make for an interesting additional feature. A third improvement would be to allow for multiple levels of nested circuits within each other. This could have been accomplished through the abstraction of the menu system to functions outside of the main function, and recursive methods.

[1] A. F. B. Santos, G. P. Duggan, C. D. Lute, and D. J. Zimmerle, An efficiency comparison study for small appliances operating in dc and ac in minigrids, 2018 IEEE Global Humanitarian Technology Conference (GHTC) , 1 (2018).

[2] T. A. Obagade and S. Konyeha, Development of alternating current (ac) line monitoring device for power system management, Pure and Applied Physics **9**, 17 (2020).

[3] A. Agarwal and J. Lang, *Foundations of Analog and Digital Electronic Circuits (The Morgan Kaufmann Series in Computer Architecture and Design)* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005).

[4] C. Gross and T. Roppel, *Fundamentals of Electrical Engineering* (CRC Press, Florida, USA, 2012).

[5] P. Horowitz and W. Hill, *The Art of Electronics*, 3rd ed. (Cambridge University Press, USA, 2015).