COMPUTER -AIDED CIRCUIT ANALYSES: WITH EMPHASIS ON GRAPHICAL SCHEMATIC ENTRIES

BY

HENRY OLABOWALE AJERE

92/2431

DEPARTMENT OF ELECTRICAL/COMPUTER ENGINEERING

FEDERAL UNIVERSITY OF TECHNOLOGY

MINNA

THIS PROJECT IS SUBMITTED TO THE DEPARTMENT OF ELECTRICAL/COMPUTER ENGINEERING IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF BACHELOR OF ENGINEERING DEGREE.

MARCH, 2000

Dedication

This project is dedicated to God almighty, one who has been my closest companion and work mate. Even amidst problems, mistakes and upheaval, He left me no doubt He is beside me.

Acknowledgements

I wish to acknowledge the following people for their supports and encouragements:
Mr. Paul Attah my able supervisor, for his patience and attention. My late father, Pa Aaron Ajere of blessed memory. My mother, Mrs. Esther Adeboye gave me all I ever needed to execute this project. I'm indebted to her. Mrs. Sipora Daoudu my sister for initiating my education. Adebola Kokumo, my fiancée for her remarkable efforts in checking my workaholic excesses. I also wish to acknowledge all my sisters, Mrs. Funmilayo Isijola, Mrs. Fatima Abdulrahaman, Mrs. Abidemi Omoyele, Abosede Ajere, Kemi Adeboye, Bukky Adeboye, Sola Adeboye, Abimbola Adeboye and little brother Mathew Adeboye. Also acknowledged is my step father Mr. Oyebiyi Adeboye, my step mother Mrs. Rebecca Adeboye, my step brother and 'friend' Rotimi Adeboye , Funmilayo Adeboye and all the entire members of Adeboye's family. I'm grateful to all my friends, Adeolu Tella and Bukola Ogunrole for their words of encouragement, Lamide and Funke, are such two wonderful friends. I also thank Jide for helping in some drawings. Finally, this acknowledgement would not be complete if a friend is not mentioned here, Mr. Ajiboye Johnson, such a kind and lovely man. He printed this report. May God reward him abundantly. (Amen)

Abstract

This report presents the software design and development of Computer-aided Circuit Analyses (CACA) program emphasizing graphical schematic entries. The software captures the designer's circuit schematic, which is drawn via the Graphical User Interface (GUI) provided by the program, and subsequently performs either AC or DC analyses as specified.

The program, is designed to run on any win32 based operating software such as window9x, windows NT x.0 etc. It was developed with the GUI and Object-Oriented Programming (OOP) features of Microsoft Visual C++ 6.0 to simulate the generation and solution of Modified Nodal Admittance Matrices (MNAM).The matrix equations are solved using LU factorization method to determine the nodal voltages and unknown currents.

The results generated for both AC and DC analyses are presented as tables relating input frequency vs. output voltage and input voltage vs. output voltage respectively.

The program will analyze circuits consisted of passive components.

TABLE OF CONTENT

Chapter One

INTRODUCTION

Background

Computer- aided design is just beginning to gain more recognition among the Nigeria academic and industrial communities. Recently, seminars and workshops were organized by various bodies and organizations across the countries, to enlighten the professionals on the technology. Among such seminars are the one organized by the Nigeria Society of Engineers (NSE) on the importance of engineering design to technological development of the nation in September 1999. The seminars emphasized CAD as a major tool for effective realization of engineering design goals. Nigeria technological advancement rests on her ability to take proper advantage of this technology. The successes recorded by the technological advanced western and Asian countries are attributed to the use of CAD systems in their design and manufacturing processes (Feu, 1988).

Today, in Nigeria, there are few PC-based CAD tools in the market: AutoCAD Release 14, the current AutoCAD 2000 and plotters etc. Some organization, institutions, government agencies, and industries are known to be making effective utilization of these tools. However, most of these CAD tools make more impact on the mechanical and architectural designs. The same cannot be said in the electrical and electronic designs. One limitation or the other impairs the few CAD tools in this field. For example, SPICE (Simulated Program with Integrated Circuit Emphasis) versions, which are common in Nigeria today, are those based on the outdated DOS platform. Many versions still rely on coded commands for describing circuit connections.

Win32 based operating system have been designed to exploit the powers and features of modern PC. Therefore many applications have been produced, which run credibly well on this platform. CAD software tools, which success relies on their abilities to integrate other tools, should not be exempted.

Recently, there are increasing number of desktops, which run win32 based operating systems in the country.
Hence, a CAD tool, which is designed for this platform, is highly recommended. This explains the reasons why this project is embarked upon.

1.0 Problems Against Computer Aided-Circuit Analysis

Recently, circuit design has progressed to the point where the design process is totally dependent on Computer Aided Design (Bushnell, 1988). Increasing number of CAD applications has since been on the market. CAD application like Designer's Workbench (DWB), Palladio, USC Expert Synthesis System, Schema, Demeter and Electric etc recorded some successes in their respective applications.

Although credible achievements have been recorded in the use of some of these tools, in the area of schematic capture, advanced method of analysis, worst-case analysis, etc. Majorities have not been designed to run on Win32 based operating environments. This has restricted their uses in Nigeria where Win32 based environment is far gaining recognitions. Any CAD tool produced for win32 will be widely used in this part of the world. This attests for the popularity of AutoCAD (a CAD tools based on win32 platform) in Nigeria. However, it (AutoCAD) has not been utilized properly for circuit design and analysis.

1.0.1. Purposes and aims of project:

To this end, the project aims to:

(a) Initiate a win32 based Computer Aided Circuit Analysis (CACA) software emphasizing schematic circuit entry.

(b) Produce menu, toolbar and toolbox driven GUI (Graphical User Interface).

(c) Makes the software extensible and easy to integrate with other tools.

(d) Exploit win32 features and power.

(e) Demonstrate the modern software development approach for engineering solution.

1.2 Significance of the Project

Win32 based CAD for electrical/electronic design is very scanty despite the overwhelming achievements recorded with other applications. This project will increase significantly, the CAD solutions to most basic circuit design problems.

In addition, CAD awareness among the Nigerian students studying electrical/electronic engineering is improved. The project can also be useful in university for studying circuit analysis. If the project is further improved, it can become a commercial product generating economic values.

1.3 Limitation of the Project

Modern software has been simplified to make even the most complex application less difficult to develop. This is made possible through the use of advanced tools and technologies in software development. However many application developed today was a result of teamwork among many developers. It is almost impossible to develop any commercial software product by one developer. For example, it has been reported that Windows NT 4.0 was produced by a team comprising well over 130 software developers across a network (Microsoft cooperation 1998). Apart from this, the time taken to design, test and deploy the software amount to 2years. These are the kind of efforts and time required to produce a commercial software product.

To this end, this project does not intend to produce such a tasking product, considering the fact that it was carried out by a single undergraduate student. As stated in the aims and objective, the project only aims at initiating a work in this regard. Hence, the software produced is limited to academic communities and subject to further improvement in the future.

Since the software is based on win32 operating system, it can only be run on PC operating Windows '95, 98; NT, OS/2, and other win32 based operating system.

In addition, the software was designed for LINEAR-PASSIVE Circuits. And the types of analyses performed are restricted to AC and DC.

Chapter Two

Literature Review

In chapter I, a number of principles and concepts underlying the project were mentioned. The main objective, as pointed out, was to initiate the development of computer aided circuit analysis in win32-based environment. In this chapter attempt will be made to further explain the concepts introduced in chapter I in order to determine how this project fits into current trend in CAD solutions. In view of this, the discussion in this chapter will focus on the following:

- Concept of design and analysis.
- Circuit design and analysis.
- Types of analysis.
- Trend in the modern circuit design.
- Computer Aided Design and Analysis.
- Concept of Modern software and programming tools.

2.0 Concept of Design and Analysis

Cognitive scientists believe that design is –exclusively- a task performed by human brain. It is defined as the creative ability to formulate alternative solutions to a problem. To this end, design is seen as a cognitive activity of human beings (Feu, 1974). It involves human ability to think, decide, lists, find, relate, obtain and suggest etc. Usually, in engineering, a problem is formulated, then the expected outcomes are specified (specification) and series of solutions are proposed in the bid to offer best design alternative. Therefore, in all the fields of engineering, design is seen as one most important activity.

Having formulated series of alternatives, the next task of course is to select the best from the series of alternatives. In doing these, each alternative is analyzed and evaluated to decide if it meets the specifications. Most of the time this process is iterative in nature and there are always predefined methods for doing it. There may be mathematical models, theories, laws and soon, which are applied in other to analyze an alternative. For example, in circuit analysis, applications of numerous circuit theories, laws and models is a common practice. Therefore, analysis of most engineering designs involve iterative and mathematical processes. As can been seen analysis is a tool, which is used to chose, test and refine the best design alternative. And a design process precedes it.

2.1 Circuit Design and Analysis

2.1.1 Circuit Design

When designing a circuit there must be a target output. A target output might be any of the following:

1. *Block diagram.*
2. *Logic diagram.*
3. *Schematic diagram.*
4. *Wiring diagram.*

Each of these diagrams utilizes symbols that represent the component or package circuits that make up the overall circuit being designed. The symbols are then connected by straight lines in such a way as to describe the circuit.

Block diagram

Block diagrams do not completely describe an electrical circuit but do indicate the complexities of the circuit, what it is supposed to do, and its main features. The used block diagrams are almost entirely simple rectangles.

Logic diagram

Logic diagrams are similar to block diagrams, in that they describe the logical progression of the electrical signals in a circuit as they go from one component or logic element to another. However, the symbols take forms that are unique to a particular function, and the diagrams are used only for digital control circuits.

Schematic diagram

Schematic diagrams describe the picture of an electrical circuit in further detail, supplying enough information for complete analysis of the circuit. Standard symbols are used and complete electrical connections are shown in such a way that the circuit can be most easily understood.

Wiring diagram

A wiring diagram is an actual picture of the circuit as it appears on a completed assembly. It is used not for analysis or explanation of a circuit but as an assembly drawing in the shop.

Each of this as said earlier, is an output of any circuit design process. Therefore it can be said that circuit design aims at producing any of the above diagrams based on the problem specifications.

Whatever the targeted output diagram, Bushnell (1988) reported that design of a circuit (especially VLSI) is inherently an iterative activity. Designers might wish to generate an initial design that is correct in all aspect, so that latter redesign would not be necessary. Unfortunately, designers cannot foresee all the consequences of their high-level design decisions and when they initially make those choices, therefore some choices will lead to incorrect design.

The output diagrams produced as a result of these design process are used for various things ranging from: explaining or describing the circuits, analyzing the circuits, and assembling the circuits. For example the circuit block diagrams are mainly used in explaining the circuits complexities at higher level of abstraction, wiring diagrams are for circuit assembling and schematic diagrams are used for circuit analyses (Machover, 1986).

2.1.2 Circuit Analysis

Circuit analysis involves application of various circuit theories to determine circuit desired parameters such output current, voltage etc. As noted earlier, circuit analysis depends on the schematic output diagram produced at the end of design. And since most circuit designs are known to be iterative, circuit analysis become inseparable from circuit design. When designer designed a schematic diagram based on the problem specification, the schematic is *analyzed* to test if it meets design specification. If does, the design is completed, if it doesn't, the circuit is redesigned until the specification are met. Bushnell (1988) agreed that circuit analysis is part of designed process. However, they are separate

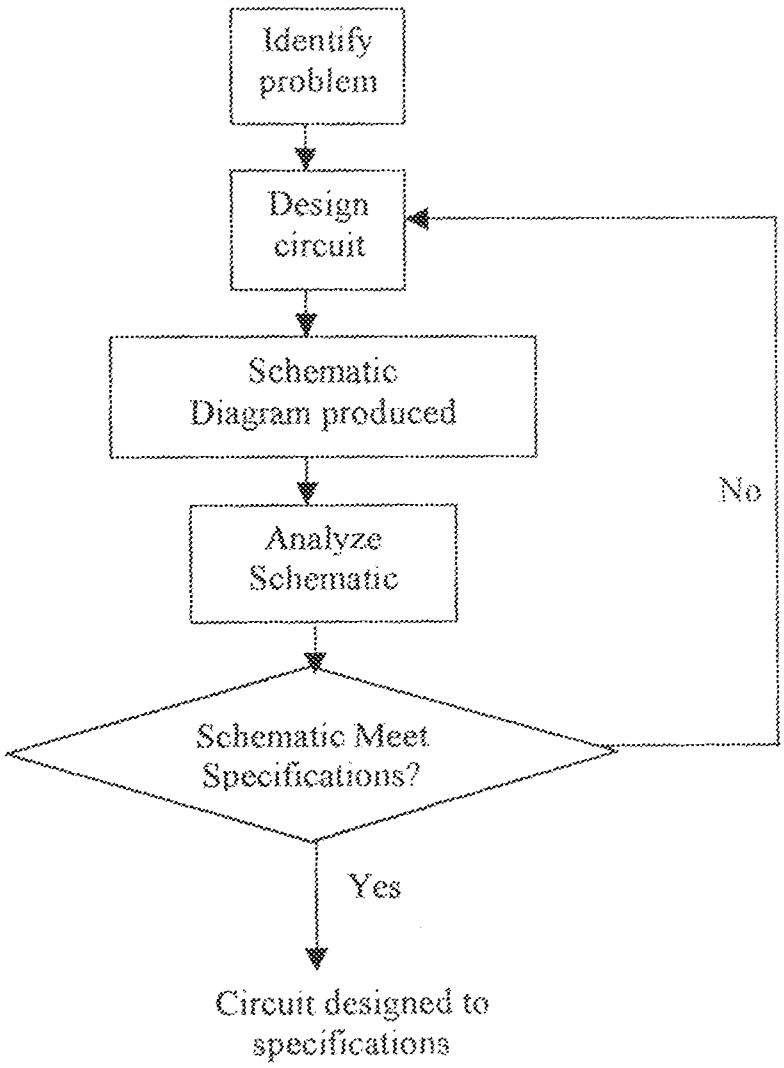


Fig2.3 Iterative Design

processes in which one complements the other.

Types of Analysis

In electrical or electronic circuit analysis, there are four basic types of analysis:

(i) Operating point

(ii) DC

(iii) AC

(iv) Transient

Operating Point Analysis: - As the name suggest, operating point analysis is used to determine the parameter value (current, voltage, power etc) of the circuit when no input signal- except for the power supply- is applied. It is sometimes called bias point. It directly affects the gain and linearity characteristics of the circuit

DC Analysis: - This measures or determines output characteristics of circuits as the amplitude of the input signal is varied. For example, a circuit might be design in way that its output voltage should not exceed 100V. The schematic design can be analyzed to verify this circuit specification. Schematic can be redesigned, or components values adjusted to correct this. Usually what is done is to plot varied input values with its consequent outputs.

AC Analysis: - This determines the output characteristics as the frequency of the input signal varies. This type of analysis is common in communication application, where frequency responses of circuits are vital.

Transient Analysis: - This is sometimes referred to as time invariant analysis. It measures the circuit response to time. For instance a circuit might be expected to gain a particular output voltage after the input signal is applied.

2.2 Trend in circuit design

Years back, circuit designers have to contend with the necessity of building breadboards, fit them with worst-case or limit devices (i.e. active device whose characteristics were at the high or low specification limits), and then see whether the circuit performed satisfactorily. The profession has come a long way then, and over the last few decades the complexities of circuits has made it quite difficult to adapt this method to design large-industrial circuits. This increased complexity is due both to advanced in technology and to the need for meeting a number of simultaneous design specifications (Fink, 1984).

This complexity has forced circuit designer to turn to computer. Today, there are Computer-aided design, Computer-aided circuit analyses and Computer-aided circuit manufacturing packages, which are used for circuit design, analyses and manufacturing.

In the future, Machover (1982) predicted the increasing availability of EXPERT programs, which are essentially software mimics of the way an expert analyzes a problem and reach a decision. This is closely associated with artificial intelligence. Such features, Machover (1982) predicted, will characterize future CAD/CAM system.

2.3 Computer-Aided Circuit Design and analysis

Computer-aided circuit design is simply the use of computer to achieve circuit design objectives. As a specific example of the use of computer as a design tool, consider the design of VLSI circuit. During the course of the design process, an engineer might use:

a) One computer program to model the 2-dimensiional effect of semiconductor devices.

b) Another computer to model the fabrication process for manufacturing integrated circuit to determine the process of variation.

c) A logic simulator to verify the logical operation of the design.

d) A layout program to help with the placement of the many thousands of the transistors of the VLSI circuit on a chip.

e) And a circuit simulator to determine the actual electrical functions of a circuit.

All these processes are automated in a design environment, and the speed and accuracy to which they are achieved has made the VLSI circuit design a realistic.

Out of these processes, (e) remains the target of this project; computerized determination of actual electrical functions of a circuit, refer to as Computer-aided circuit analysis.

It was mentioned earlier that circuit analysis is the determination of circuit desired output parameters. However, to analyze a circuit, simultaneous equations describing the circuit have to be generated by applying appropriate circuit theories. Consequently the equations are solved to obtain the desired output. From this, two things are obvious in analyzing a circuit:

1) Generation of simultaneous equations describing the circuit.
2) Solutions to the equations.

For a computer, numerical method is the preferred option than analytical method. And the followings are required of computer-aided circuit analysis software:

(i) Algorithm for describing the circuit schematic.

(ii) Algorithm for generating simultaneous equations from the schematic.

(iii) Algorithm for solving simultaneous equations.

In this project, (i) was achieved by allowing schematic description through diagrams. Though there are many techniques used in formulating set of equations describing an electronic network, the technique employed in this project is the *Nodal Approach* (the theory behind this approach is explained in the next chapter). Finding a numerical solution requiring the solution of a set of simultaneous linear (or nonlinear) equations for a DC solution, and a set of simultaneous nonlinear differential equation for a transient solution forms the core of any circuit analysis program. To this end, a matrix method, which is found suitable for programming will be used in this project.

2.4 Modern software and programming tools

Not very long ago, large-scale mission-critical enterprise applications were the exclusive province of massive mainframe computers (MSDN, 1998). That's changing rapidly. This change is due to the rapid increase in hardware sophistication and improvement. Software industries quickly took advantage of this.

By this, Ergonomics became the center focus in software production. A leading software industry, Microsoft software Inc, took the initiative by first launching its GUI-oriented (Graphical User Interface) operating system by early 90s (Microsoft corporation, 1998). Prior to this, the corporation was known for its popular TUI-oriented (Text User Interface) DOS operating system. With this operating software, a User types his/her commands at the *command prompt*. However, because of the difficulty in learning to use these commands, and the needs to make computer systems more *User-friendly*, the company released windows3.0. This was later updated to windows3.1, 3,11 3.12 etc. However, Windows 3.x is made to work dependently on DOS, which makes it to be an expansion of DOS rather than an operating system itself. Clamors for an independent operating system led to the release of Windows95, followed by windows98, the current windows operating system. Microsoft has announced windows2000, which is due for in release February 2000 (http://www.msn.com/microsftnews, 1999).

Improvement in the operating software is always greeted with improved application software. Hence the advent of windows operating software led to the release of series of application like word perfect for windows, office95-2000, Dbase for windows, to mention just few. All these are characterized by GUI, object-oriented features, OLE etc.

Modern programming tools.

Windows applications are made easier to use due to improvement in the operating software, likewise the tools for developing these applications. Following are some of the tools used in developing some applications:

(i) Visual Basic
(ii) Visual C/C++
(iii) Visual Java
(iv) Visual FoxPro

These tools deviate from the tradition of relying on codes as means of instruction while focusing on WYSIWYG (What You See Is What You Get) approach of coding system. For instance, a programmer might want to draw a rectangle in its application. Using the traditional approach, the programmer issues series of commands by typing codes, which are susceptible to bugs. However, this task can be easily performed in the modern programming tools by simply using a pointing device (mouse). This task is performed in seconds. In the next chapter, some techniques in modern programming are considered in details.

2.5 Summary

In this chapter an attempt has been made to discuss design, analysis and the computer method of achieving the goals (of design and analysis). It was explained that the output of any design work is a diagram specifying solution to the problems at hand. Specifically, circuit design aims at producing different circuit diagrams: schematic diagram, logic diagram, and wiring diagram. Schematic diagram is the main input required for circuit analysis. Set of simultaneous equations is generated from the schematic diagram.

Consequently, the equations are solved to obtain required output parameter. The use of computer in doing this was also explained.

Modern software and programming tools open a new way of developing highly complex program like CAD. The introduction of modern operating system coupled with sophisticated programming tools changes programming from the old code-oriented to visual oriented approach.

3.1 Matrix Analyses of network

Employing Thevenin-Norton theorem or star-delta transformation, followed by combination of series and parallel combination, can reduce the intricases in circuit analysis. However computer cannot be programmed for such task. Hence computer needs a general predetermined method to perform analysis.

3.2 Modified Nodal Formulation

The Modified Nodal Approach (MNA) is a hybrid equation formulation method that allows voltage and current variable to be unknowns. That is, unknown currents and voltages are part of the equations generated. In fact, if circuit contains only linear conductance and independent current sources, the MNA reduced to the nodal equation $YV_n = J$, where Y is the nodal admittance matrix, V_n is the node voltages (excluding the earth) and the J is the current source vector. By allowing the unknown current variables in the MNA equations, the MNA is able to accept all the four types of controlled sources, the independent current and voltage sources, any type of non-linearity and linear R, L, C elements.

By considering Kirchoff's laws at each node, the modified nodal equations are generated (i.e. the current leaving a node is zero). However, to consider current as unknown requires the branch relationship (*BR*) of the element to be added to the set of node equations. The result for linear network, is a set of equations of the form

$$\begin{bmatrix} Y_r B \\ C D \end{bmatrix} \begin{bmatrix} V_n \\ I_b \end{bmatrix} = \begin{bmatrix} J \\ F \end{bmatrix} \text{ ---------- } \quad \text{Equation 3.1}$$

the independent current and voltage sources, any type of non-linearity and linear R, L, C elements.

By considering Kirchoff's laws at each node, the modified nodal equations are generated (i.e. the current leaving a node is zero). However, to consider current as unknown requires the branch relationship (*BR*) of the element to be added to the set of node equations. The result for linear network, is a set of equations of the form

$$\begin{bmatrix} Y_r B \\ CD \end{bmatrix}\begin{bmatrix} V_n \\ I_b \end{bmatrix} = \begin{bmatrix} J \\ F \end{bmatrix} \text{---------} \quad \text{Equation 3.1}$$

where,

Y_n = (n - 1) X (n - 1) is the nodal admittance matrix,
B = (n- 1) X (n -1) matrix taking account the certain unknown branch current leaving the node.
C = b X (n -1) matrix expressing certain branch relationships e.g.: voltage source relationship
D = b X b matrix accounting for certain controlled source branch relationships,
J, F = (n -1)- and b-dimensional vectors, respectively, which are corresponding right-hand-side (RHS) entries.
n = no of nodes in the circuit

The modified nodal equations are easily formulated by considering a *stamp* for each element in the circuit.

Indicating the K_{th} column of Yr and C by V_k, the K_{th} column of B and D by I_k, the K_{th} row of Yr and B by V_k, and the K_{th} row of C and D by I_k. For instance, if the K_{th} branch is a conductance G_k, and it is connected between nodes i and j, and the current through the conductance is not a desired output variable, the *stamp* would be

3.1 Matrix Analyses of network

Employing Thevenin-Norton theorem or star-delta transformation, followed by combination of series and parallel combination, can reduce the intricacies in circuit analysis. However computer cannot be programmed for such task. Hence computer needs a general predetermined method to perform analysis.

3.2 Modified Nodal Formulation

The Modified Nodal Approach (MNA) is a hybrid equation formulation method that allows voltage and current variable to be unknowns. That is, unknown currents and voltages are part of the equations generated. In fact, if circuit contains only linear conductance and independent current sources, the MNA reduced to the nodal equation $YV_n = J$, where Y is the nodal admittance matrix, V_n is the node voltages (excluding the earth) and the J is the current source vector. By allowing the unknown current variables in the MNA equations, the MNA is able to accept all the four types of controlled sources, the independent current and voltage sources, any type of non-linearity and linear R, L, C elements.
By considering Kirchoff's laws at each node, the modified nodal equations are generated (i.e. the current leaving a node is zero). However, to consider current as unknown requires the branch relationship (*BR*) of the element to be added to the set of node equations. The result for linear network, is a set of equations of the form

$$\begin{bmatrix} Y, B \\ CD \end{bmatrix} \begin{bmatrix} V_n \\ I_b \end{bmatrix} = \begin{bmatrix} J \\ F \end{bmatrix} \qquad \text{——— Equation 3.1}$$

(a) The current through the specific element is desired as an output.

(b) Or there are other elements in the circuit, either controlled sources or non-linearities that depend on the current.

Given a general node below, the *stamps* for the elements are generated. Table3.3 Shows *stamps* for various elements connected to the node.

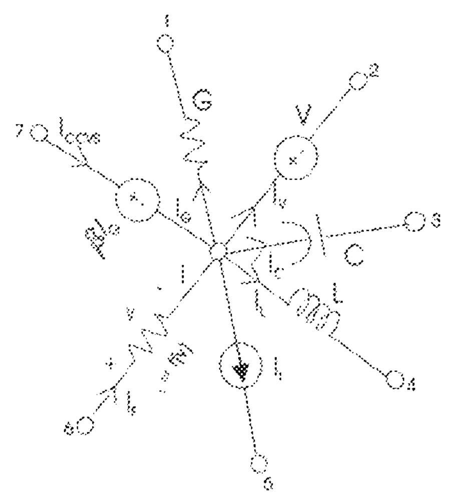
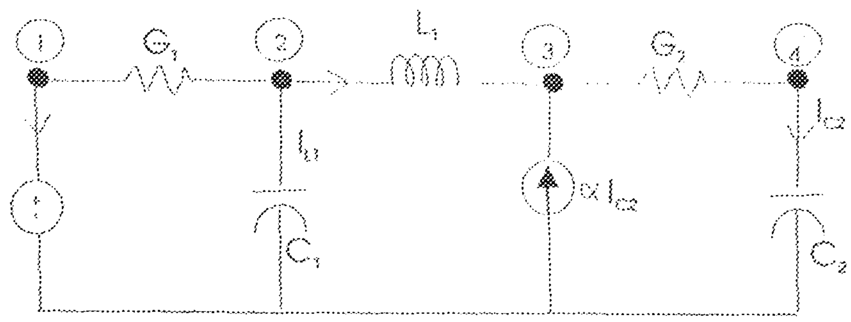


Fig 3.1 General to define stamps for element

Fig 3.2 Circuit example



Having defined stamps for elements, an attempt will now be made to apply them to the circuit above:

Applying the stamp for each element, the following matrix equation are generated:

$$\begin{bmatrix} G_1 & -G_1 & 0 & 0 & 1 & 0 & 0 \\ -G_1 & G_1 + C_1 \frac{d}{dt} & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & G_2 & -G_2 & 0 & -1 & -\alpha \\ 0 & 0 & -G_2 & G_2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & -L\frac{d}{dt} & 0 \\ 0 & 0 & 0 & C_2 \frac{d}{dt} & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ I_v \\ I_L \\ I_{C2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ V \\ 0 \\ 0 \end{bmatrix}$$

Matrix equations generated.

<table>
<tr><td colspan="2">lement type current not output</td><td>Branch current is output</td></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>V_i</td><td>V_1</td><td>RHS</td></tr>
<tr><td>V_i</td><td>G</td><td>-G</td><td></td></tr>
<tr><td>V_1</td><td>-G</td><td>G</td><td></td></tr>
</table>
</td><td>
<table>
<tr><td></td><td>V_i</td><td>V_1</td><td>I_G</td><td>RHS</td></tr>
<tr><td>V_1</td><td></td><td></td><td>1</td><td></td></tr>
<tr><td>V_1</td><td></td><td></td><td>-1</td><td></td></tr>
<tr><td>I_G</td><td>G</td><td>-G</td><td>-1</td><td></td></tr>
</table>
</td></tr>
<tr><td></td><td></td><td>
<table>
<tr><td></td><td>V_i</td><td>V_1</td><td>I_G</td><td>RHS</td></tr>
<tr><td>i</td><td></td><td></td><td>1</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>-1</td><td></td></tr>
<tr><td></td><td>1</td><td>-1</td><td></td><td>V</td></tr>
</table>
</td></tr>
<tr><td></td><td>
<table>
<tr><td></td><td>V_i</td><td>V_1</td><td>RHS</td></tr>
<tr><td>V_i</td><td>C d/dt</td><td>-C d/dt</td><td></td></tr>
<tr><td>V_3</td><td>-C d/dt</td><td>C d/dt</td><td></td></tr>
</table>
</td><td>
<table>
<tr><td></td><td>V_1</td><td>V_3</td><td>I_C</td><td>RHS</td></tr>
<tr><td>V_i</td><td></td><td></td><td>1</td><td></td></tr>
<tr><td>V_3</td><td></td><td></td><td>-1</td><td></td></tr>
<tr><td>I_C</td><td>C d/dt</td><td>-C d/dt</td><td>-1</td><td>V</td></tr>
</table>
</td></tr>
<tr><td></td><td></td><td>
<table>
<tr><td></td><td>V_1</td><td>V_3</td><td>I_C</td><td>RHS</td></tr>
<tr><td>i</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>1</td><td>-1</td><td>-L d/dt</td><td></td></tr>
</table>
</td></tr>
</table>

(V)

	V_i	V_6	RHS
V_i	$-F(V_6-V_i)$		
V_6	$F(V_6-V_i)$		

	V_1	V_6	I_F	RHS
V_1			-1	
V_6			1	
I_F	$F(V_6-V_i)$		-1	

I

	V_i	V_5	RHS
V_i			-I
V_5			I

	V_1	V_5	I_I	RHS
V_1			1	
V_5			-1	
I_I			1	I

CCV

	V_1	V_7	Iccv	I_G	RHS
V_1			-1		
V_7			1		
I_G	-1	1		$-\beta$	

Table 3.3 Stamps definition for elements

The example given above demonstrates the use of MNA to generate set of equations, which sufficiently describe a circuit. Next is the methods employed in solving the equations.

Solution of the simultaneous equations.

There are many methods for solving matrix equations. Among the methods are the following:

1. Direct solution
2. Gaussian matrix reduction.
3. LU factorization.

LU factorization, a method employed in this project will be discussed.

LU factorization method of solving simultaneous equations

LU factorization is a numerical procedure used in factorizing a square matrix into the product of lower triangular matrix **L** and an upper triangular matrix of **U**.

Considering a square matrix **A**, where **A** is to be the product **LU** of a lower triangular matrix **L** in which the leading diagonal element are unity and an upper triangular matrix **U** such that

$$[A]=[L][U] \text{ ———— Equation 3.3}$$

There are n^2 unknown coefficients of **L** and **U** and also n^2 equations in the expansion of equation (3.x). In the method of triangular decomposition these equations are in sequence to obtain all the unknown coefficients.

The main use of triangular decomposition is at the first stage of the solution of a set simultaneuos equations of the form **AX** = **B**. The whole solution process comprises three stages as follows:

Stage1: Matrix **A** is factorized into **LU**. (**A** = **LU**) *(Factorisation)*

Stage2: Solution of matrix **Y** is obtained by forward substitution in

LY = **B**. *(Forward Substitution)*

Stage3: Solution of matrix **X** is obtained by backward substitution in

UX = **Y**. *(Backward Substitution)*

Here **Y** represents an intermediate variable. Although the last two stages both involve the solution of simultaneous equations, their computational requirements are low because of the triangular form of their coefficient matrices.

Certain features make the triangular decomposition computationally easy:

a) The unknown coefficients L and U, can be merged to give an *n x n* matrix

$$A^F = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ l_{21} & u_{22} & \cdots & u_{2n} \\ . & . & . & . \\ l_{n1} & l_{n2} & \cdots & u_{nn} \end{bmatrix} (= L + U - 1)$$

b) If solved in row-wise or column –wise order the elemental equation each yields just one unknown per equation. In general, the equation of a_{ij} can be used to determine either l_{ij} or u_{ij}, whichever is unknown, as shown by the underlined variables in the following 3 x 3 decomposition:

$$\left\{\begin{array}{llll} a_{11}=u_{11} & a_{12}=u_{12} & \cdots & a_{1n}=u_{1n} \\ a_{21}=l_{21}u_{11} & a_{22}=l_{21}u_{12}+u_{22} & \cdots & a_{2n}=l_{21}u_{1n}+u_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1}=l_{n1}u_{11} & a_{n2}=l_{n1}u_{12}+l_{n2}u_{22} & \cdots & a_{nn}=\sum_{k=1}^{n-1}l_{nk}u_{kn}+u_{nn} \end{array}\right\}$$

The formulae for computing element $\mathbf{A}^F$ are:

$$l_{ij}=\frac{a_{ij}-\sum_{k=1}^{j-1}l_{ik}u_{kj}}{u_{jj}}(j<i) \qquad \text{Equation 3.4}$$

$$u_{ij}=a_{ij}-\sum_{k=1}^{i-1}l_{ik}u_{kj}(j\geq i) \qquad \text{Equation 3.5}$$

c) It is possible to overwrite **A** with the elements of $\mathbf{A}^F$ as they are formed,thereby reducing the computer memory used in the computation.

d) The computation of each element may be carried out with double precision arithmetic by using an inner product routing.

e) In the forward and back-substitution operations the matrix B may be overwritten in turn by Y and then X.

f) The operation on the right-hand side represented by stages 2 and 3 can be carried out independently of the decomposition (stage1).

Capatilising on these features, the following algorithm is used for triangular decomposition.

Algorthm for Triangular Decomposition

```
1.  FOR I = 1 TO N
{
        {
            FOR J = 1 TO N
                X = A (I, J)
                JJ = J-1
                FOR K = 1 TO JJ
                {
                        X = X-A (I, K) * A (K, J)
                }// END THIRD LOOP

        }//END SECOND LOOP

}//END FIRST LOOP
```

A pratical implementation of this algorithm will include an error message and escape sequence to be triggerred when a zero diagonal element is encountered.

To perform the stage2 backsubstitution, the following agorithm is implemented:

Algorithm to perfom Backsubstitution

There are *m* right-hand sides

```
FOR IR = TO N
{
        I = N-IR+1
        II = I+1
        FOR J = 1 TO M
        {
                X = B (I, J)
                FOR K = II TO N
                {
                        X = X-A (I, K) * B (K, J)
                }//END THIRD LOOP
                B (I, J) = X/A (I, I)
        }// END SECOND LOOP
}//END FIRST LOOP
```

A successful implementation of triangular decomposition depends on the divisor being non-zero at every stage. However, MNA produces well- conditioned matrices, which may not encounter zero diagonal element.

3.3 Programming fundamentals and basics.

3.3.1 Object-Oriented Programming (OOP)

The world is made up of both physical objects and concepts. To effectively model or simulate the real world and human processes in code, it would be good if software could be made up of representations of these real-world objects. That is exactly what object-oriented programs attempt to do.

Objects are designed to represent real 'objects' and concepts in codes. Here are some examples of these real 'objects':

- A Component
- A Resistor
- A Current
- A Branch
- A Node

All of these are objects in the physical world, and the whole idea behind using software objects is to create representations of these same objects inside software applications. Then application can make these objects interact with each other just as they would in the physical world. For instance, *current* flows in and out of the *Node*.

Traditional programs are centered around the processes and procedures that the program is trying to model. Object-oriented programs, on the other hand, are centered around the real-world objects that the program is trying to model. Once these objects have been identified and modeled, one can write programs that use these objects to simulate the processes and procedures in a much simpler and cleaner way than other programming approaches would accomplish. In this project, as will later be shown in the next chapter, many of such objects are modeled such as: component, resistor, inductor, capacitor etc.

3.3.2 Object Linking And Embedding (OLE)

OLE is a mechanism that allows users to create and edit documents containing items or "objects" created by multiple applications. If application A produce "Aobj" object, and application B produces "Bobj" object. Another application C can be used to edit these objects as if they belong to it. This foster application integration and reduces the burdens of programming. For instance a circuit CAD application might be developed that produces circuit schematic diagram. Another CAD developed to perform circuit analyses. The later can call the object produced by the former, work on it without any data conversion.

3.3.3 Class Library

This comprises sets of object classes, which can be accessed to perform various tasks. Microsoft Foundation Class library (MFC) comprises of classes which define various object used in this project. For instance, *Device Context*, a class that encapsulates windows graphical functions, is part of the MFC library.

3.3.4 Dynamic Linking Library (DLL) Vs Static Linking Library

A dynamic-link library (DLL) is an executable file that acts as a shared library of functions. Dynamic linking provides a way for a process to call a function that is not part of its executable code. The executable code for the function is located in a DLL, which contains one or more functions that are compiled, linked, and stored separately from the processes that use them. DLLs also facilitate the sharing of data and resources. Multiple applications can simultaneously access the contents of a single copy of a DLL in memory. In static linking, the linker gets all the referenced functions from the static link library and places it with code into executable file. Using DLLs instead of static link libraries makes the size of the executable file smaller. If several applications use the same DLL, this can be a big savings in disk space and memory.

3.3.5 C++ Document/View Architecture.

By default, MFC applications (application based on the Microsoft Foundation Class library MFC) use a programming model that separates a program`s data from the display of that data and from most user interaction with the data. In this model, an MFC document object reads and writes data to persistent storage. The document may also provide an interface to the data wherever it resides (such as in a database). A separate view object manages data display, from rendering the data in a window to user selection and editing of data. The view obtains display data from the document and communicates back to the document any data changes.

Chapter Four

Design And Implementation

The last chapter delved into some theoretical concepts underlying the project. Modified Nodal Admittance (MNA) matrices (theoretical backbone of the project), LU factorization methods for solving simultaneous equation etc were among the various topics treated.

In this chapter, the processes and steps that led to the development and subsequent implementation of the project will be explained in stages. Hence the chapter covers the following:

- Stage1: Preliminary
- Stage2: Implementation and Testing

1.1 Stage1: Preliminary

Because planning constitutes a major step in software development, the program developed in the course of this project undergone rigorous planning exercise. The aim of this exercise is to setup a proper framework, which would guide the development and the subsequent determination of what are required in implementing the program. Therefore, consistent with this, followings steps were embarked upon in a top-down design approach:

4.1.1 Step1: Identifying Program requirements

The requirements were obtained from the *scenario* described below:

> Circuit designer, using user-friendly interface, enters the schematics – comprising of conventional components symbols, nodes and interconnecting lines- into the system.
> The user specifies/ inputs the types of analyses he wanted. The system later captures, validates, and interprets the schematic to generate the system specific data (set of simultaneous equations) required to fully analyze the circuit. Solving the equations the results are presented to the designer in a format chosen (tabular/graphical forms).

From the scenario described above it is required that:

1. User interface allows for schematic entries.
2. There must be mechanism for schematic entries
3. The system must be able to capture the schematic.
4. There must be mechanism for interpreting the schematic
5. There must be mechanism for converting schematic to the system specific data (set of simultaneous equations). In other words the system should generate the simultaneous equations required to analyze the circuit.
6. There are types of analyses.
7. There are different output formats that can be chosen.

4.1.2 Step2: Identifying Objects.

OOP (Object-Oriented Programming) was described in the last chapter as programming approach, which tends to model-in code- physical objects and their interaction, as they would appear in real world. In the requirements enlisted above, there are functionalities and *objects*, which can be modeled as programming entities. Therefore, in this step, attempt is made to identify the objects. The approach used in identifying the objects is to examine the words- in the scenarios and the requirements outlined-that are *nouns*. Every noun in these is potentially capable of being an object. To this end, the following were identified as the prospective objects:

I. Circuit designer.
II. User-Friendly interface.
III. Circuit Schematic.
IV. Component.
V. Symbol.
VI. Connecting Lines.
VII. Nodes.
VIII. System.
IX. System Specific Data (MNAM)
X. Results.
XI. Output format.

4.1.3 Identifying Object Relationships

Since objects interact with each other, relationships exit between them. Just as objects represents nouns in the scenario and the enlisted requirements, the functionalities that describe their (objects) relationships are identified by examining *verbs* that seem to connect them. Hence the following relationships are obtained:

I. Designer **Uses** the UI
II. Designer **Enters** schematic.
III. System **Captures** schematics.
IV. System **Validates** schematics.
V. System **Interprets** schematics.
VI. System **Generates** MNAM.
VII. System **Solves** MNAM
VIII. System **Analyzes**
IX. UI **Gets** the output format.
X. UI **Outputs** results.
XI. Component **Is-a** Schematic
XII. Nodes **Is-a** Schematic.
XIII. Connecting Line **Is-a** Schematic.
XIV. Resistor **Is-a** Component.
XV. Inductor **Is-a** Component.
XVI. Capacitor **Is-a** Component.

4.1.4 Step3: Modifying The Objects and Relationships

Clearly, some of the possible objects listed won't end up being modeled as objects in the software. For instance *designer* object would not need to be modeled, hence it is removed from the lists.

Finally, objects attributes and behaviors are defined. The diagram below summarizes what was later obtained in the process.

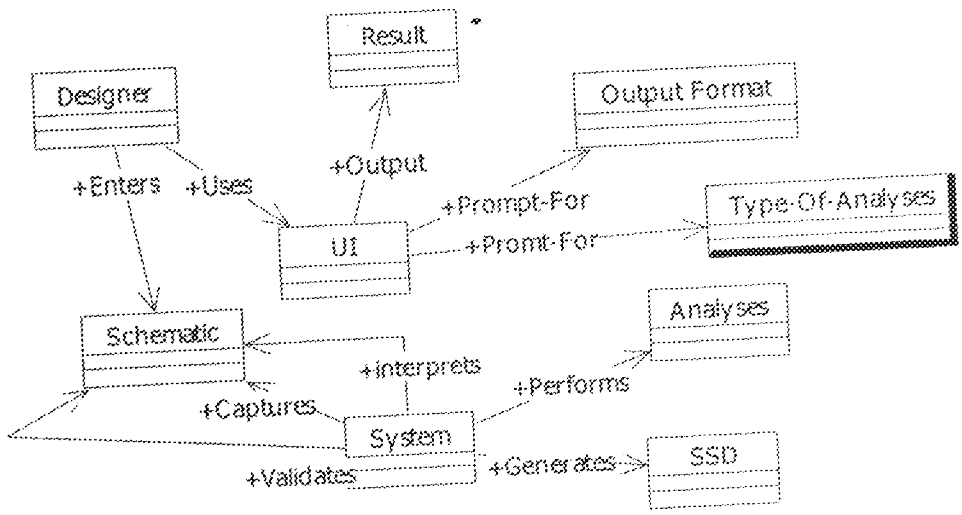


Fig 4.1 Objects, attributes, behaviors and relationships

1.2 Stage2: Implementation

The previous stage provides a framework that produces the objects, object attributes, objects behaviors and object relationships that are required in the codes. In this stage, their software implementations are described. It is also necessary at this stage to state the choice of the programming tools used for the implementation.

Visual C++ is widely regarded as a true object-oriented programming tool because of the ease it employed in implementing the three fundamental features of OOP namely:

i. Encapsulation/Aggregation

ii. Inheritance.

iii. Polymorphism.

Since space would not permit to describe these features, any interested reader is hereby refered to books, which explain the detailed concepts. One of such book is the '*Professional Visual Basic 5.0 Business object*'.

However, because the features are very important in implementing the objects and functionalities obtained in the course of this project is the rational behind the choice of visual C++ as the programming tool. Precisely, Microsoft Visual C++ 6.0 (part of Microsoft Visual Studio 6.0) is used.

Applying the Microsoft Foundation Class (MFC) application framework and various features in this version of MS VC++ the following summarizes the steps undertaken to implement the design requirement obtained in the previous stage.

4.2.1 Step1: Creating Skeleton Application

AppWizard- an application generation wizard provided by the framework- creates the files for a skeleton application, including source files for application, document, view, and frame windows; a resource file; a project file; and others — all tailored to specifications.

AppWizard was run and the options needed were specified in the options pages. Options include making the application a COM component, container, or both; adding Automation; and making the application database-aware. In this project, the following options were chosen:

i. Document/View Architecture
ii. Multiple Document Interface (MDI)
iii. ActiveX Control
iv. Docking Toolbar
v. Initial Status bar
vi. Print/Print Preview
vii. 3D Controls
viii. MFC Standard
ix. File Extension (*.cct)
x. Use MFC as a shared DLL

AppWizard integrate these options into the skeleton application generated. See appendix C for the classes and files that were automatically included.

4.2.2 Step2: Building and running the skeleton application

From the visual c++ Integrated Development Environment, otherwise known as 'workspace', the *Project//Execute Circuits.Exe* command are clicked to build and run the skeleton application.

The running skeleton application derives many standard File, Edit, View, and Help menu commands from the framework. For this project, which is an MDI application, fully functional Window menu were included and the framework manages creation, arrangement, and destruction of MDI child windows.

4.2.3 Step3: Modifying the resources generated in the skeleton application

The default resource file created by AppWizard supplies many of the resources needed; Menu resource, toolbar resource, icon resource and bitmap resource. In this step, some of these resources were modified and additional resources added. Precisely, IDR_CIRCUITYPE (a menu resource), and IDR_MAINFRAME (a toolbar resource) were modified in the Visual C++ resource editor.

To IDR_CIRCUITYPE, the following were added:

a). *Analyse*, pop-up menu with DC (its resource id is ID_ANALYSE_DC) and AC (its resource id is ID_ANALYSE_DC) menu commands.

b) *Orientation*, pop-up menu with cascaded menu commands; Horizontal, Vertical pop-up menus.

To IDR_MAINFRAME toolbar, the following buttons were added:

a) A button with the an id number ID_ANALYSE_AC (shortcut to the Analyse//AC menu command.
b) A button with the an id number ID_ANALYSE_DC (shortcut to the Analyse//DC menu command.
c) A button with an id number ID_DELETEAC menu command.

An extra toolbar was also created (it is called toolbox). This contains commands for creating circuit components. In Addition to the ***CAboutDlg*** (IDD_ABOUTBOX) box generated in the skeleton application, two extra dialog boxes were created:

a) ***Ccompvaldlg***(IDD_COMPONENT_VALUE), which accepts a component value from the user.

b) ClsourceDlg (IDD_SOURCE_VALUE), which accepts the values of a source from the user.

The modifications made to the toolbars and menu bar need *handler functions* (functions to be executed when a user via the user interface sends any of the commands to the program). However, the functions are delayed until the document objects are created.

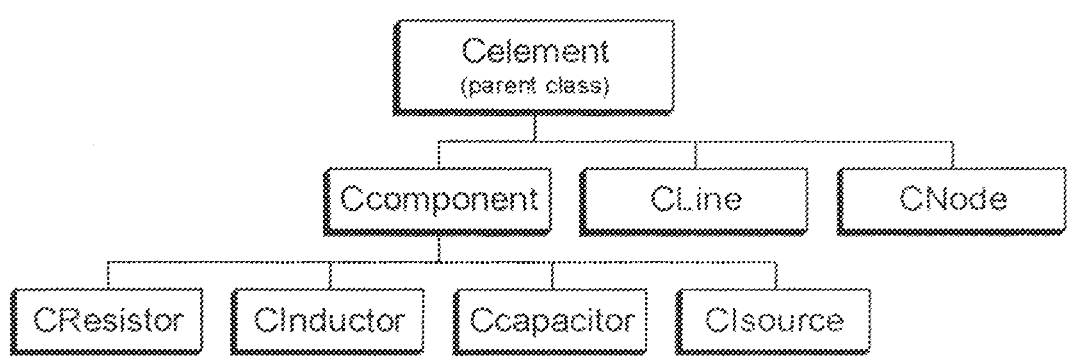
4.2.4 Step4: Creating Additional Classes

This step utilizes the products of stage2. Fortunately, objects such as *System* and *UI* have been implemented in the skeleton application generated by AppWizard. Infact, the System object was implemented by the framework *document/view* architecture. This architecture contains *CcircuitDoc* (for document) and *CcicuitView* (for view) classes. The *UI* was implemented by the framework windows objects such as *CchildFrame* (client window) and *CmainFrame* (application window).

The remaining objects: Schematic, Component, Resistor, Inductor, Capacitor and the Source were created with their implementations listed in the appendix.

However, the chart below summarizes their organization as maintained in the code.

Component Chart



4.2.5 Step5: Implementing Document class

The CcircuitDoc class implemented in the skeleton application is a Document class. However, modification is necessary to customize to suit the project requirements. To state precisely, the following modifications were added in the course of implementing this class:

Variables:

a) m_elementList: - a C++ template collection CTypedPtrList object, which contains the elements that describe a circuit.

b) m_tracker: - a CRectTracker objects, which provides a visual feedback when an element is selected.

c) m_coeff: - a CArray (2-dimensional array) object, which is the coefficient matrix of the MNAM.

d) m-matrhs: - a CArray (1-dimensional array) object, representing the right-hand-side vector matrix of the MNAM.

Functions:

For the functions implemented in the document class refer to the CcircuitDoc.h header file in the appendix.

The framework already knows how to interact with document data files. It can open and close document files. However, to read and write the document's data, and handle other user interfaces, the document Serialize function was implemented accordingly. See appendix for the listing.

4.2.6 Step6: Implementing the view class

Just as the CcircuitDoc class modified, the CcircuitView class was also modified to customize the skeleton application. To this end, followings were added:

Variables:

a) m_pSelected: - a pointer to Celement object. It points to the address location of the element currently selected.

b) M_pCreated: - a pointer to Celement object. It points to the memory address location of a newly created element.

Functions:

The functions implemented in this class can be found in the CcircuitView.h file of the appendix.

Finally, it must be stated here that at each step describe above Building, Compiling, Debugging and Running were performed. However, the way these were carried out was earlier explained in step2 when building the skeleton application.

Chapter Five

Testing, Results and Discussion Of Results

The last chapter explains the processes and the implementations that led to the software developed in this project. This chapter aims at explaining the testing and the subsequent results obtained. This is done to evaluate the extent to which the project initial objectives have been met.

5.1 Testing and Results

Running the debug version of the application (Circuit.exe), the circuit below was inputted and component values entered accordingly.

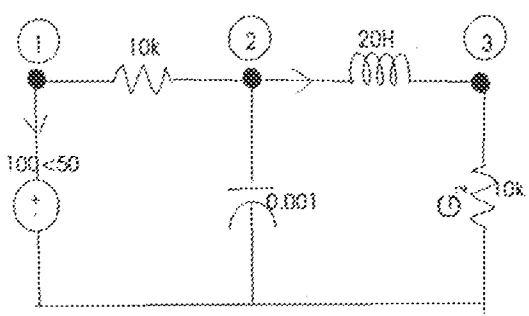


Fig 5.1 Example Circuit Tested.

Performing AC analyses for frequency range of 0-50hz step 10, the following table summarizes the results obtained.

Freq	Vout
0.0	#DIV/0!
10.0	-3.399760907307E-004+6.6940350736855E-004i
20.0	2.61535813194046E-005-9.55221510409382E-005i
30.0	1.71071194696105E-006-1.686060202059E-005i
40.0	-4.67155106227535E-006+4.11161331026123E-005i
50.0	3.71767931905035E-006-3.82421600617977E-005i

Table 5.1 Results Obtained For AC analyses

The same circuit was tested for DC analyses. Given an input voltage range of 0-100V (magnitude) and keeping the frequency constant at 50hz. The table below summarizes the results.

Input Voltage	Vout
0.0	-1.03952413947598E-007-1.03910400703171E-008i
20.0	6.60373932651994E-007-7.65674484441582E-006i
40.0	1.42470027925158E-006-1.53030986487613E-005i
60.0	2.18902662585117E-006-2.29494524531068E-005i
80.0	2.95335297245076E-006-3.05958062574523E-005i
100.0	3.71767931905035E-006-3.82421600617977E-005i

Table 5.2 Results Obtained For DC analyses

Analyzing the same circuit manually (by applying circuit rules and theory) yields the following results for both AC and DC (using the same values as previously analyzed by circuit.exe).

Freq	Vout
0.0	0.0
10.0	-3.10E-007+6.390E-004i
20.0	2.416E-005-9.332E-005i
30.0	1.610E-006-1.38E-005i
40.0	-4.00E-006+3.99E-005i
50.0	3.61E-006-3.72E-005i

Table 5.3 manual results for AC

Input Votltage	Vout
0.0	-1.67E007-1.67-008i
20.0	5.901E007-6.23E-006i
40.0	1.59E006-1.5103E-006i
60.0	1.8E006-3.10E005i
80.0	2.56E006-3.500E-005i
100.0	3.71E006-3.9200E-005i

Table 5.4 manual results for DC

5.2 Discussion of the results

The results show that:

a) The graphical user interface produced in the program allows easy and graphical schematic entries. Specifically, a designer enters his circuit descriptions by using the simple click-drag-drop features of windows OS.

b) Comparing the results obtained manually and ones obtained by using the circuit.exe shows some variations, which may be due to: memory overflows, and round off errors. Nevertheless, the results are still viable.

Chapter Six

Conclusion and Recommendation

The main aim of this project as stated in chapter one is to produce computer-aided circuit analysis software for win32 application platform while emphasizing graphical schematic entries. The nodal analysis method was adopted specifically to generate the Modified Nodal Admittance Matrices (MNAM), which describes the circuit. Subsequently, the LU factorization was used to solve the set of equations.

The processes above were programmatically modeled (in codes) using the object-oriented features of OOP.

To this end, this chapter aims to:

- access the extent to which the objectives were realized,
- state and explain the problems encountered in the course of developing the software,
- state the program shortcomings and limitations, and
- recommend further improvements, which can be made to extend the capabilities of the program.

6.1 Project Assessments and Conclusion

There are two areas in accessing the project, thus: areas of *schematic entries* and *analyses*.

In the area of schematic entries, graphical implementation has been achieved. All the circuit conventional symbols were used to describe the schematic and this has been very successful.

However, in the area of analyses, the program recorded some abilities to handle passive circuits to a larger extent. Nine out of fourteen circuits tested yielded correct results. Three circuits produced wrong results when tested while two failed completely. The anomaly noticed in some circuits may be due largely to runtime bugs, which were not discovered during the software development.

By and large, it can be concluded that the project has credibly achieved its main objectives.

6.2 Problems encountered in the course of development.

Paramount of all the problems encountered is lack of adequate information. There is no material in the library on Win32 programming. This sets the work back a great deal as valuable time was spent looking for materials.

6.3 Shortcomings, Limitations and Recommendation.

The following are the shortcomings discovered in the program:

1. The User Interface (or the program desktop) has no visual feedback to indicate the object selected from the application desktop.

Normally the button should stay depressed until the selected component is drawn.

2. When AC menu commands is pressed twice in a section, the debug version of the application asserts. This is a bug that needs to be corrected.

The applications is limited- as rightly pointed in chapter one- to a passive circuit elements. However, it can be extended by adding active circuit elements such as VSCC (Voltage Source Current Controlled) CSVS (Current Source Voltage Controlled) etc, which can easily be used to modeled transistors, diodes and soon.

The shortcomings and the limitation are generally due to analyses aspect of the work. More accuracy will be achieved if User Interface of the application (circuit.exe) is integrated with the SPICE program. SPICE can run in the background as the circuit analyzer while the application converts schematic entered graphically into SPICE commands.

-Appendix-

CircuitView Header

```
// CircuitsView.h : interface of the CCircuitsView class
//
/////////////////////////////////////////////////////////////////////////////

#if
!defined(AFX_CIRCUITSVIEW_H__73FCF20E_99BA_11D3_A4A7_A74622EA4837__INCLUDED_)
#define AFX_CIRCUITSVIEW_H__73FCF20E_99BA_11D3_A4A7_A74622EA4837__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


class CCircuitsView : public CView
{
protected: // create from serialization only
	CCircuitsView();
	DECLARE_DYNCREATE(CCircuitsView)

// Attributes
public:
	CCircuitsDoc* GetDocument();
	//CRectTracker m_tracker;

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CCircuitsView)
	public:
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	virtual void OnInitialUpdate();
	protected:
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnUpdate(CView* pSender, LPARAM lHint, CObject* pHint);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CCircuitsView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif


	Celement* m_pSelected;
```

-Appendix-

```
// Generated message map functions
protected:
	CRect SetRectangle(CPoint pt);
	void SetupTracker(CRectTracker& tracker, Celement* pItem);
	void DoSelection();
	void SetSelection(Celement* pNewSelection);
	Celement* HitTestItem(CPoint& pt);
	Celement* m_pCreated;

	//{{AFX_MSG(CCircuitsView)
	afx_msg void OnMouseselect();
	afx_msg void OnNode();
	afx_msg void OnResistor();
	afx_msg void OnLine();
	afx_msg void OnInductor();
	afx_msg void OnCapacitor();
	afx_msg void OnCurrentsource();
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	afx_msg void OnOrientationHoriLeft();
	afx_msg void OnUpdateOrientationHoriLeft(CCmdUI* pCmdUI);
	afx_msg void OnOrientationHoriRight();
	afx_msg void OnUpdateOrientationHoriRight(CCmdUI* pCmdUI);
	afx_msg void OnOrientationVertDown();
	afx_msg void OnUpdateOrientationVertDown(CCmdUI* pCmdUI);
	afx_msg void OnOrientationVertUp();
	afx_msg void OnUpdateOrientationVertUp(CCmdUI* pCmdUI);
	afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in CircuitsView.cpp
inline CCircuitsDoc* CCircuitsView::GetDocument()
  { return (CCircuitsDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_CIRCUITSVIEW_H__73FCF20E_99BA_11D3_A4A7_A74622EA4837__INCLUDED_)
```

CircuitDoc Header

```
// CircuitsDoc.h : interface of the CCircuitsDoc class
//
/////////////////////////////////////////////////////////////////////////////
#include <complex>
```

-Appendix-

```
#include <afxwin.h>
#include <afxtempl.h>
#if
!defined(AFX_CIRCUITSDOC_H__73FCF20C_99BA_11D3_A4A7_A74622EA4837__INCLUDED_)
#define AFX_CIRCUITSDOC_H__73FCF20C_99BA_11D3_A4A7_A74622EA4837__INCLUDED_


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

using namespace std;

class CNode;

struct CELL // declaring the cell type
	{
		UINT column, row;
		complex<float> value;
	};
struct RHS // for the right hand side
{
	UINT row;
	complex <float> value;
};
```

Cstamp Header

```
/////////////////////////////////////////////////////////////////////////////
// CStamp class

class CStamp: public CObject
{
	DECLARE_SERIAL(CStamp)
	//Construction
public:
	CStamp(); // empty constructor for serialization
	//Attributes
	UINT I,J,K;
	CArray <CELL, CELL&> m_cells;
	CArray <RHS, RHS&> m_rhs;



	//Operations
	//Overrides
	virtual void Serialize(CArchive &ar);

 };
```

-Appendix-

Cendpoint Header

```
//////////////////////////////////////////
//class name:Cendpoint

class Cendpoint : public CObject
{
public:
	// constructor
	Cendpoint(); //empty constructor
	Cendpoint(CPoint pt); // copy constructor


	DECLARE_SERIAL(Cendpoint)
	//Attributes
	CPoint m_point;
	enum enumpolar{NEGATIVE, POSITIVE} m_polar;
	UINT m_nNodeNumberConnectedWith;
	enum enumstate {CONNECTED,NOTCONNECTED} m_state;
	CNode *m_pNodeConnected;

	//Operation
	void connect();
	void disconnect();

	virtual void Serialize(CArchive &ar);
	//Implementation

};//ends Cendpoint
```

Celement Header

```
//////////////////////////////////////////

// Celement declaration
// See the implementation in CircuitDoc.cpp

class Celement : public CObject
{
	// constructor and initialization
	//DECLARE_DYNCREATE(Celement)
public:
	Celement(); // standard constructor. Has to be empty for serialization
	DECLARE_SERIAL(Celement)

	//Attributes
```

»

-Appendix-

```
        CSize m_size; // diagonal size of an element rectangle
        CRect    m_symbolrect;
        //Operations
//public:
        virtual void Setup();

        virtual bool DrawSymbol(CDC *pDc);
        // bool MoveRectPos(CRect *pNewRectPos);
        virtual void Serialize(CArchive &ar);

        //Implementations
};// ends Celement declaration
```

Ccomponent Header

```
////////////////////////////////////////
//
// Class:Ccomponent
// Base class:Celement
// electrical components inherits from it:
//          starts Ccomponent declaration:
class Ccomponent : public Celement
{
        // Constructor/destructor
public:
        Ccomponent();
        DECLARE_SERIAL(Ccomponent)
        virtual void Setup();
        // Attributes
        Cendpoint m_firstpoint;
        Cendpoint m_secondpoint;
        CStamp m_stamp;



        enum unit{ ohms,henrys,farads,amps } m_unit;
        enum Orientation { HORIZONTAL, VERTICAL } m_orientation;
        enum Direction {LEFT, RIGHT, UP, DOWN} m_direction;

        float m_flval; // the component value
public:

        //Operations
        bool SetVal();
        //bool SetEndPoint();
        void Setsize();
protected:
        bool Vdown(CDC *pDc, Cendpoint& pt1, Cendpoint& pt2);
        bool Vup(CDC *pDc, Cendpoint& pt1, Cendpoint& pt2);
        bool Hright(CDC *pDc, Cendpoint& pt1, Cendpoint& pt2);
        bool Hleft(CDC *pDc, Cendpoint& pt1, Cendpoint& pt2);
```

```
        bool Draw2terminal(CDC *pDc, Cendpoint& pt1, Cendpoint& pt2),
        DivideRectFour(LPRECT bigrect, LPRECT smallrects);
        DrawArm(CDC *pDc, POINT from, POINT to);
        virtual bool Draw1stbody(CDC *pDc, LPRECT prect);
        virtual bool Draw2ndbody(CDC *pDc, LPRECT prect);
        //Implementations
public:
        float m_frequency;

        bool virtual DrawSymbol(CDC *pDc);
        virtual void Serialize(CArchive &ar);
        virtual void SetStamp(BOOL bAsVar = FALSE, UINT k = 0);
}; //ends Ccomponent declarations
```

CResistor Header

```
////////////////////////////////////////////
// class name:CReisitor
// Base class: Ccomponent

class CResistor : public Ccomponent
{
public:
        //constructor and destructor
        CResistor();// for serialization
        virtual void Setup();
        DECLARE_SERIAL(CResistor)
        ~CResistor();
        //Attributes

        // Operation


        // Implementation

        bool DrawSymbol(CDC *pDc);
protected:
        bool Draw1stbody(CDC *pDc, LPRECT prect);
        bool Draw2ndbody(CDC *pDc, LPRECT prect);
public:
        virtual void Serialize(CArchive &ar);
        virtual void SetStamp(BOOL bAsVar = FALSE, UINT k = 0);
};
```

CInductor Header

```
////////////////////////////////////////////
//class name:CInductor
//base class:Ccomponent

class CInductor : public Ccomponent
```