Imperial College London

Department of Electrical and Electronic Engineering

ELEC40006: Electronics Design Project

Circuit Simulator Technical Report

Authors

Xin Wang
CID: 01735253
xin.wang19@imperial.ac.uk

Brandon Cann
CID: 01724765
brandon.cann19@imperial.ac.uk

Adam Rehman
CID: 01720256
adam.rehman19@imperial.ac.uk

Submitted in partial fulfillment of the requirements for ELEC40006

June 5, 2020

Contents

1 Overview of the report

This report loosely follows the stages of the Software Developement Cycle.

Section 2 details the design problem presented and the program requirements at are necessary for a preliminary design to be established.

Section 3 provides a summary of the program development timeline and the details relating to project management format the format of meeting minutes used to how the responsibilities are distributed among the project team.

Section 4 discusses the preliminary designs produced by the team and the rationale behind some design choices implemented by the team.

Section 5 gives an comprehensive overview of the program design, detailing the functions and flowcharts used.

Section 6 investigates the design constraints of the program, the accuracy of the results produced and the speed of execution as the circuit size varies.

Section 7 serves as a continuation of Section 6, discussing the possible improvements to mitigate problems encountered during testing.

Section 8 details the possible features that can be added on to the existing program and the required modifications to the program to add the mentioned features.

2 Project Specification

2.1 Design Problem

Develop a program that is able to read in a file describing a circuit specified by the user, perform transient simulation on that circuit and output the calculated voltages at each instance in time into a file.

2.2 Program Requirements

The main program requirements are listed as follows:

- Program must support basic circuit components listed as follows [1]:
  - Resistors
  - Ideal Capacitors
  - Ideal Inductors
- Input file must adhere to SPICE netlist formats
- The output file must be in Comma Separated Value (.CSV) format.
  - Columns of output file represent nodes in the circuit.
  - Rows of output file represent an instance in the simulation.

[1] Advanced component can be supported provided development schedule is not constrained.

2.3 Design Criteria

The team has identified several factors from the list given in the Product Design Specification document.

- Maintenance: One of the most important factors the team identified. As the development timescale is constrained and the features required is open-ended, so it is important we ensure the design can incorporate new features efficiently and cost-effectively if the client wishes to add new features.

- Documentation: For a non-intuitive program, proper documentation is required for client and future programmers to make modifications should the client wish.

- Performance: No strict performance guidelines are given by the client but the execution speed should still be reasonable.

- Time scale: 6 weeks with a definite deadline. It is important to balance proper team management techniques and creating a program that meets client standards.

- Testing: Numerous test programs have been created to test various aspects of the program to ensure accurate results have been produced. The results produced have been verified with LTSpice, a well established circuit simulation program.

- Patents: SPICE engine is a public domain software so we could possibily utilise the engine.

- Ergonomics: A Graphics User Interface (GUI) is not specified by the client and could possibly make the program much easier to use.

3 Team Management

3.1 Project timeline overview

TIMELINE 1: *Final Project Timeline*

12 May	Team formed.
14 May	1^{st} team meeting Tech Specifications published.
15 May	Research and design of preliminary designs.
16 May	First version of project management guidelines established.
20 May	2^{nd} team meeting.
21 May	Final version of project management guidelines established.
29 May	Final program design agreed on.
3 June	3^{rd} team meeting.
4 June	Final program design implemented.
6 June	Technical Specification published.
14 June	Project deadline

Documentation and minor improvements on codebase is a constant process throughout this process.

3.2 Gnatt chart

3.3 Management approach

The project team, with research on different forms of project management, decided on using a Waterfall project management approach. The phases are listed as follows:

- System and Software Requirements.
- Analysis.
- Design.
- Coding.
- Testing.

This method is selected mainly since the project is short, requirements are clear and the team is not constantly required to report to the client at the early stages of the software development to ensure client statisfaction.

The Waterfall methodology prioritises proper documentation throughout the whole development process. This is important to the team as the team project requires a Project Report to be submitted and, due to the ongoing remote lab orals and family-related obligations during quarantine, allows a team member to be quickly caught up should a member step out of the team for a while.

The disadvantage of the Waterfall methodology lies mainly in that no working software is produced until late in the cycle so there is a level of risk and uncertainty with not being able to meet the deadline, especially in the current state of the world [2]. The project team later realised the methodology is not exactly suitable for the Object Orientated Approach that the team eventually settled on. As the integration is done at the end, there is a possibility that any errors in the program is not caught during integration.

[2]Any unforseen circumstances are listed in team meeting minutes summarised the following sections.

3.4 Team responsibilities breakdown

The responsibilities of each team member is discussed during the 1st meeting and formalised in the 2nd meeting. The respective roles are partly determined by the Belbin questionnaire provided by Mrs. Perea [3]

The main responsibilities of each team member is listed as follows:

- Adam Rehman
  - Plant
    - Research program.
    - Coding the program.
    - Testing program.
- Brandon Cann
  - Monitor Evaluator
    - Ensure documentation managed by Xin Wang and coding managed by Adam is in sync.
    - Contributes to program aspects that is falling behind.
    - Ensure project requirements are met.
- Xin Wang
  - Implementer
    - Creating and managing the team Project Report and various other documentations.
    - Meeting minute keeper.
    - Ensuring code written is up to a standard format with proper comments.
    - Manages repository base.

[3] Belbin questionnaire forms are found in Appendix: Belbin Roles.

3.5 Project meeting minutes

The meeting minutes follow a standard template that is attached under Appendix: Meeting Minutes.

There is no standard protocol for calling team meetings, team meeting are called usually when there is a problem that is applicable to all team members. Usually, there is informal communication across various channels from WhatsApp to Discord.

Over the course of the development cycle, there has been three formal meetings and the main reasons and conclusions are listed below.

- Meeting 1
  - Reason:
    - Project team meeting each other. Not every team member knows each other.
    - Discuss the Circuit Simulator briefing.
  - Conclusions:
    - Xin Wang was assigned as documentation manager and tasked with formalising the Project Requirement.
- Meeting 2
  - Reason:
    - With a draft of the project management guidelines, team discussed the team dynamics.
    - Discussed general direction of our program design.
  - Conclusion:
    - Agreed that practicing proper team management is just as important as showing practical skills in coding.
    - Final team management guidelines are agreed and tasked to Xin Wang to formalise in documentation.

- Meeting 3
  - Reason:
    - Program basic requirements are nearly fulfilled.
    - Meeting to discuss future program design direction.
    - Xin Wang has a notification by South African Embassy of a Repartiation flight.
  - Conclusion:
    - Plans set to divide up Xin's responsibilities among the other two team members.
    - Documentation finalised and notes written by Xin passed to team members.

4 Preliminary Designs

Our initial program design relied on the project briefing provided by Dr.Stott. Parsing a input file into a data structure is a standard procedure. The main point of discussion is related to how the program solves for unknown voltages.

The main design choices taken by the team has been set out below.

4.1 Object Orientated Programming

As the number of circuit components that we need to support is open ended besides the mandatory basic components according to the Project Breifing, the team was concerned on the expandability of the program.

The OOP approach allows the program design to be rapidly and cost-effectively altered to accomodate new features such as additional circuit components or analysis techniques [3] that the client might request later on in the program's lifecycle. This approach saves the programmer and the client cost and time in program maintainability.

The disadvantage to that approach is that it has shown to be slower than other preliminary programs designed without OOP to an average of of 5% to 10% slower depending on other optimisations in the program.

It is the team's view that execution speed was not the highest priority and the ability to expand was also important. The design is detailed in the next section.

4.2 Modified Nodal Analysis

Modified Nodal Analysis (MNA) is an extension of normal Nodal Analysis. When the team start researching similar programs like PSPICE and LT-Spice, MNA was mentioned numerous times [1] and its benefits was clearly seen when we tried to write a program that implemented normal Nodal Analysis. A problem the team encountered was trying to represent the current-dependent circuit elements like inductors efficiently.

As stated earlier, a primary concern of the team lies in the scalability of the program. By using MNA, just like other commercial programs, the team is confident that the analysis component of the program will not be a bottleneck to future developments of the program.

THe advantage of MNA is in its standard procedures. The standardised algorithm for solving nodal equations allowed the team to modularise the solving aspect of the program. This meant not constantly have to make changes to one of the most important parts of the program, instead just ensuring the component is in the compatible format to be entered into the MNA algorithm.

The MNA algorithm consists of three matrices to form the matrix relation: $Ax = b$:

- A is consists of four submatrices which includes the admittance matrix. The program treats matrix A as a whole matrix instead of separating it into four small matrices and combining at the end.
- x represents the unknown node voltages and current of the voltage sources.
- b contains the values of voltage sources and current sources in the circuit.

The admittance matrix is created during the initialisation of the simulation. Non-linear elements are converted into suitable formats at each instance in time during a transient simulation. The resulting parameters are 'stamped' into the matrix at each instance. Only 'stamping' the changing values instead of creating a new system of matrices saves valuable execution time during program execution. The matrix inversion is handled by the Eigen library, the library supports various types of inversion with difference accuracy and speed and this is discussed in the Optimisation sector.

4.3 Eigen Matrix Library

In terms of matrix operations, the team decided to use a third-party program to handle the matrix-related operations. The primary concern with choosing a third-party program was how seamless the program integrates into our program. The team does not know whether the client would be comfortable installing an unknown third-party program on their system besides the program that the client specifically requested for.

There are third-party programs that require installation such as Armadillo and some has a large file size such as the Boost Library which is close to 1GB. Based on the concerns laid out, the team looked for a program that has a reasonable size, does not require installation in order to be used. The

third-party program chosen was Eigen. It is a file that has a reasonable size and only requires that the Eigen is located in the same directory and the execution scripts we provided is used. A copy of Eigen is included in the program files so client's ease of usage.

5 Program Design

5.1 Top level view of the program

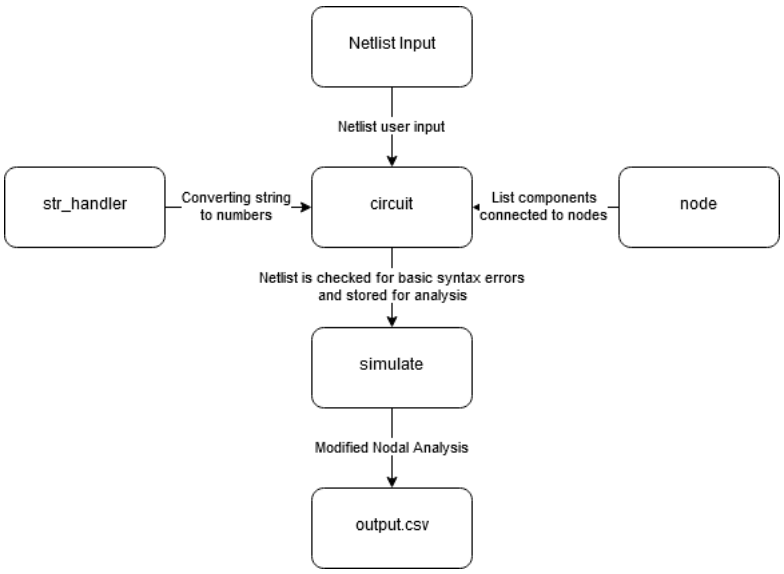


Figure 1: General flowchart of program

circuit.hpp file forms the central section of the program, handling the input of the file and storing it in a form that allows the *simulate* file to perform Transient Analysis on it and outputs the file in the format (.CSV) specified by the client.

The theoretical information and program design for each of the main files are discussed in the following sections.

5.2 Components files

Due to the OOP design methodology, each component and its necessary functions are described in its own respective file. The file *edge.hpp* is the base from which all other circuit components are derived from the **components** folder.

5.3 *str handler.hpp* file

The *str handler* file manages all string-related functions.

The main function is the is the *tokenisier* function which takes in an input sentence and separates it into individual words that can be processed, converted into the required formats for processing and stored.

During conversions, checks are performed at various stages. In the product briefing, there was no mention of error handling requirements but the team viewed that defensive programming was required and any errors encountered will be displayed to the user.

5.4 *node.hpp* file

A circuit can be expressed as a Network Graph, containing branches and edges. The *node.hpp* file expresses a node in the circuit and allows nodal equations to be created.

This function is not necessary in the final design since the program skips creating nodal equations and directly inputs entries into the matrix before solving it.

5.5 *circuit.hpp*

As mentioned previously, *circuit.hpp* manages all aspects of the program from reading in the user input, checking basic syntax errors during user entry to ensuring the simulation is successful. Numerous checks are performed over the course of the simulation and any errors encountered will instantly result in the simulation ending.

The flow chart of *circuit.hpp* is shown as below:

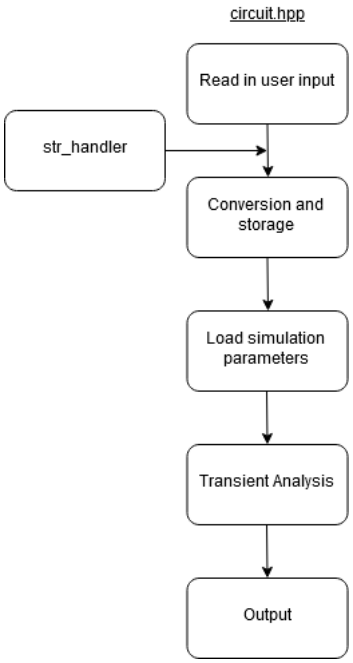


Figure 2: Circuit flow chart

5.6 *simulate.hpp*

This section of the program was where the team has spent most of the time and also where the team saw many possible optimisations which will be discussed in later sections. This section handles matrix entry and controls the transient simulation.

But using the MNA technique, basic components such as voltage sources, current sources and resistors are simple and can be directly entered into the matrices for solving. However, components such as inductors and capacitors which are non-linear devices has required further processing in order to support those components. At each instance in time, non-linear components need to be approximated into corresponding companion models. With the program using MNA, all non-linear components are converted into a current source and resistor to allow efficient entry into the matrix.

5.6.1 Non-linear components

In the process of approximating non-linear components, integration is required and there are three different methods of integration each with its own corresponding equations defining current and resistance of the companion model. Each type of integration offers different degree of accuracy which is explored under the Optimisation section.

Capacitors are defined with the following differential equation:

$$I = C \times \frac{dv}{dt}$$

Inductors are defined with the following differential equation:

$$V = L \times \frac{di}{dt}$$

Method	Parameters
Forward Euler: $I = i_n$	$V_{eq} = v_n + \frac{\Delta t}{C} i_n$ $R = 0$
Backward Euler: $I = i_{n+1}$	$V_{eq} = v_n$ $R = \frac{\Delta t}{C}$
Trapezoidal: $I = (i_n + i_{n+1})/2$	$V_{eq} = v_n + \frac{\Delta t}{2C} i_n$ $R = \frac{\Delta t}{2C}$

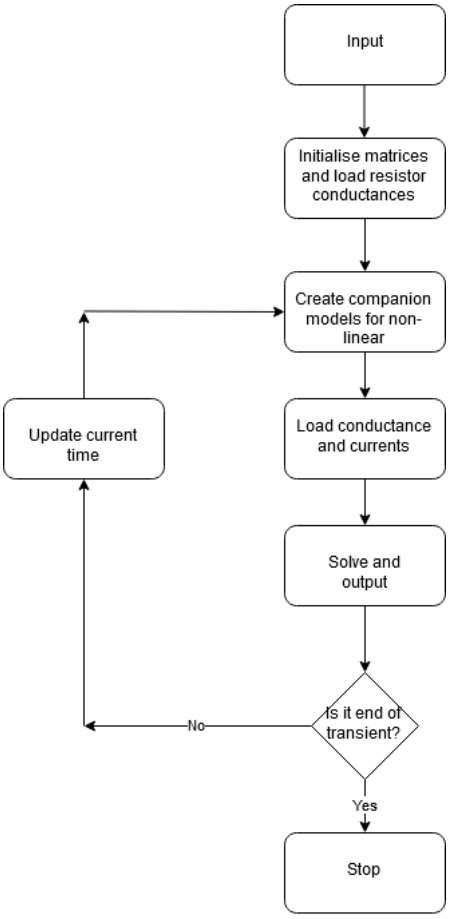
Figure 3: Capacitor companion model parameters for different types of integration [2]

Method	Parameters
Forward Euler: $V = v_n$	$I_{eq} = i_n + \frac{\Delta t}{L} v_n$ $g = 0$
Backward Euler: $V = v_{n+1}$	$I_{eq} = i_n$ $g = \frac{\Delta t}{L}$
Trapezoidal: $V = (v_n + v_{n+1})/2$	$I_{eq} = i_n + \frac{\Delta t}{2L} v_n$ $g = \frac{\Delta t}{2L}$

Figure 4: Inductor companion model parameters for different types of integration [2]

5.7 Simulation flowchart

The listed simulation flowchart has not accounted for approximating components like Diodes which need Newton-Raphson iteration.



6 Testing

It should be noted that majority of the testing is done on Linux Ubuntu 16.04.1 LTS which runs on a virtual machine in Windows 10, some testing have been completed on a Windows 10 machine. As such, the listed execution times are the average of 10 execution times.

Numerous functions are imbedded into the program itself to aide the benchmarking process and can be used to aide in debugging.

7 Optimisations

7.1 Sparse Matrices

Various circuit simulation engines like SPICE uses sparse matrices and it was mentioned [3] during our research stage but the current program used dense matrices. The primary concern was stability of the program as the team did not have any experience with Sparse Matrices but the Eigen Library is able to implement sparse matrices.

With the current supported circuit components, using dense matrices does not affect the program speed significantly. But as the circuit size gets bigger and more complicated components like operational amplifiers is supported, the matrices would get bigger. The memory used and the processing power used would increase proportionally.

Sparse matrices solves that problem as numerous entries of the matrices is zero and does not affect the results. The benefit is reduced memory usage and much faster program execution times which is a bottleneck in the current design. Due to the OOP program design, it can be quickly implemented but extensive testing would be needed.

7.2 Integration method

There are numerous different types of integration but the three most common forms of integration are as follows:

- Forward Euler
- Backward Euler
- Trapezoid

Out of the three, Trapezoid would be the most accurate form of integration but the most processor intensive form of integration listed. The report has the three forms and the defined parameters.

But Euler's methods are only accurate to the first degree while other methods such as the Runge-Kutta method is much more accurate. This method would rely on creating a specialised function that exclusively handles it and would add to the processing time but, with the implementation of sparse matrices, it would still result in a faster program.

7.3 Inversion method

The Eigen library supports numerous types of equation solver functions. The default type that the Eigen documentation supports is the *ColPivHouseholderQR* method but, using it, resulted in a consistent execution time of 50% longer. Instead, the team did not use the equation solver functions. The team used the inversion function on matrix A and multiplied it directly to matrix b.

The team believes that there is be more efficient third-party algorithms that can help make the program more efficient. For example, the Boost Library contains numerous functions that can entirely eliminate the need for the *str handler.hpp* file and, as it is a extensively supported library, it would be more efficient to use it. But it would require client consultations to ensure the client understands and is comfortable with a larger program size.

8 Adding on

8.1 Graphics User Interface

8.2 Diode

8.3 Voltage controlled sources

8.4 Current controlled sources

8.5 Operational Amplifiers

9 Appendix

9.1 Belbin Roles

Question	IMP	CO	SH	PL	RI	ME	TW	CF
1	g 1	d 3	f 0	c 3	a 0	h 0	b 0	e 3
2	a 0	b 0	e 0	g 4	c 0	d 3	f 0	h 3
3	h 0	a 6	c 0	d 4	f 3	g 0	e 0	b 3
4	d 0	h 0	b 0	e 6	g 0	c 10	a 0	f 0
5	b 5	f 0	d 0	h 5	e 0	a 0	c 0	g 0
6	f 0	c 0	g 0	a 5	h 0	e 5	b 0	d 0
7	e 0	g 0	a 0	f 0	d 0	b 0	h 0	c 10
TOTAL	6	3	0	24	3	18	0	19

Figure 5: Belbin Roles of Adam Rehman

Question	IMP	CO	SH	PL	RI	ME	TW	CF
1	g 1	d 0	f 0	c 0	a 0	h 3	b 4	e 2
2	a 3	b 0	e 0	g 0	c 0	d 2	f 5	h 0
3	h 1	a 2	c 1	d 0	f 0	g 4	e 2	b 0
4	d 0	h 0	b 0	e 0	g 4	c 2	a 3	f 1
5	b 1	f 0	d 0	h 0	e 0	a 3	c 3	g 3
6	f 2	c 2	g 0	a 1	h 0	e 1	b 1	d 3
7	e 4	g 3	a 0	f 0	d 0	b 1	h 2	c 0
TOTAL	12	7	1	1	4	16	20	9

Figure 6: Belbin Roles of Brandon Cann

Question	IMP	CO	SH	PL	RI	ME	TW	CF
1	g	d 3	f 1	c	a	h 2	b 2	e 2
2	a 2	b	e 3	g 1	c	d 4	f	h
3	h 4	a	c 3	d	f	g 3	e	b
4	d 3	h	b	e	g 2	c	a 2	f 3
5	b 3	f	d 2	h	e 3	a	c	g 2
6	f 4	c 3	g 1	a 2	h	e	b	d
7	e 4	g	a	f 2	d	b 2	h	c 2
TOTAL	20	6	10	5	5	11	4	9

Figure 7: Belbin Roles of Xin Wang

10 References

References

[1] Erik Cheever. *Analysis of Circuits*. DOI: https://lpsa.swarthmore.edu/Systems/Electrical/mna/MNA1.html.

[2] Steven Herbst and Antoine Levitt. "Companion models for basic nonlinear and transient devices". In: (2008).

[3] Taku Noda. "Object Oriented Design of a Transient Analysis Program". In: Jan. 2007.