Spice Replica Report(2022)

Ethan Sifferman

[[1]](#footnote-1)

***Abstract*—This C++ implementation of Spice (created for ECE 594BB Fall 2022 taught by Peng Li) supports DC and Transient analysis of several linear and nonlinear circuit elements including voltage sources, current sources, resistors, capacitors, inductors, N-MOSFETS, and P-MOSFETS. This tool works by Modified Nodal Analysis (MNA) and the matrix multiplication library Eigen.**

**This document explains current state of the tool by**

* **Giving reasonings for design decisions.**
* **Supplying voltage/time graphs of several circuits.**
* **Providing a list of all known issues as it stands.**

**This design is posted to GitHub under** [**https://github.com/sifferman/spice\_replica**](https://github.com/sifferman/spice_replica)**.**

# I. Running The Code

The way the code is currently organized is unintuitive. The flow to generate graphs from a “.ckt” file is to first run the MATLAB script “ckt\_to\_json.m”, which parses all specified “.ckt” files into JSON files which the C++ code can read. Second, in a C++ file, you construct a circuit object using “circuit c(<json\_file>)”. Next, you call your preferred analysis method, such as “c.tran()”. Next you run “c.to\_json(<output\_filename>)” to export the analyzed circuit to a JSON file which can be read by MATLAB. (All the C++ is provided as an example in [main.cpp](https://github.com/sifferman/spice_replica/blob/main/main.cpp)). Finally, you can run the MATLAB script “json\_plot.m” to generate the plots requested in the “.ckt” file. For the final project, I plan on making these steps more intuitive.

# II. Code Structure

So far, I have put in about 60 hours into making this tool. I began writing it in Python, but since I am more experienced in C++, and I found myself wanting more control over memory and more object-oriented-design support, I switched over to C++.

Because of the incredible object-oriented focus of C++, I spent a long time refining the class declarations and hierarchy of a circuit element. Below is a summary of the circuit header file with all the circuit elements:

|  |
| --- |
| class circuit {      struct node {};      struct mosfet {          node \*NodeD, \*NodeS, \*NodeG;          double W, L, V\_T, MU, LAMBDA, C\_OX, C\_J;          double I\_DS(double V\_GS, double V\_DS) const;      };      struct linelem {          node \*Node1, \*Node2;      };        struct resistor : linelem {          double resistance;      };      struct storage\_device : linelem {};      struct capacitor : storage\_device {          double capacitance, initial\_voltage;      };      struct inductor : storage\_device {          double inductance, initial\_current;      };      struct power\_source : linelem {};      struct V\_source : power\_source {};      struct V\_dc : V\_source {          double voltage\_value;      };      struct V\_pwl : V\_source {          voltages\_t voltages;      };      struct I\_source : power\_source {};      struct I\_dc : I\_source {          double current\_value;      };      struct I\_pwl : I\_source {          currents\_t currents;      };      std::vector<linelem\*> linelems, itrelems;      std::vector<mosfet\*> mosfets;      std::unordered\_map<int,node\*> nodes;      void dc();      void tran();  }; |

**Fig. 1.** Circuit header file summary.

Using the hierarchy described in the header file, I was able to avoid a lot of redundant code. For example, the way to calculate voltage across a linear element is to calculate the difference between the two nodes that connect it. Similarly, both capacitors and inductors need to be converted to a linear model before applying MNA, so it is helpful to group them together as “storage devices”.

# III. Libraries Used

There were three external libraries used for this design:

* The “.ckt” parser provided by Peng Li which parses a given “.ckt” file into a MATLAB struct.
* [Eigen](https://eigen.tuxfamily.org/) - Linear algebra library used for creating, solving, and manipulating matrices.
* [NLohmann JSON](https://github.com/nlohmann/json) – JSON library used for importing and exporting JSON files.

# IV. Linear DC Circuit Analysis

In getting started with linear circuit support, I found an excellent resource that explained how to build a MNA matrix for resistor circuits: <https://cheever.domains.swarthmore.edu/Ref/mna/MNA2.html>. This resource greatly helped me get started and acted as the framework for all my future MNA intuition.

To implement capacitors and inductors for DC analysis, you can simply treat each capacitor as voltage source set to the capacitor initial voltage, and treat each inductor as a current source set to the inductor initial current. Then, by solving the MNA equation, you can find the voltage at each node and current through every element.

# V. Linear Transient Circuit Analysis

Resistors circuits in MNA transient analysis are trivial since only the “z” vector changes while the “A” matrix stays the same.

However, support for capacitors and inductors are more complicated. Both need to be stamped into the MNA matrices using a Norton equivalent such that and are found upon solving the circuit.

The reason why we use a Norton equivalent as opposed to a Thevenin equivalent is to not create any extra nodes, simplifying our algorithm.

These are the equations I used to calculate and for Capacitors and Inductors:

Capacitor

Inductor

*Issue: I am not certain why these models work. I found these equations via trial-and-error. I don’t know if they use Forward-Euler, Backwards-Euler, Trapezoidal, or something else. Once I got them working, I just wanted to move on to the next parts as quickly as possible.*

Using this model, I was able to graph the following circuit behaviors:

Graphical user interface

Description automatically generated

**Fig. 2.** Voltage of a 1F capacitor, initially charged to 100V, discharging through a 1Ω resistor.

Graphical user interface

Description automatically generated

**Fig. 3.** Voltage of a 1H inductor, initially charged to 100A, discharging through a 1Ω resistor.

Chart

Description automatically generated

**Fig. 4.** Voltage across a capacitor given a series RLC circuit (1Ω, 1H, 1F) with a series voltage source (1V).

# VI. Non-Linear Circuit Analysis

Non-linear circuits bring new challenges because you do not know what model you should use to solve the circuit until you have solved the circuit. To combat this issue, we use the Newton-Raphson method for approximation.

To implement the Newton-Raphson method, you first guess and initial voltage. Then you create a Norton equivalent using and dependent on that voltage. Then you solve the circuit. Next, using the new calculated voltage drop across that element, create a new Norton equivalent and repeat until the voltage converges. (The code iterates until a change of <1e-9.)

For example, the following model works for solving circuits with diodes:

<https://www.desmos.com/calculator/fnqqrkfcby>

I initially implemented diode support as a steppingstone to MOSFETS. The current design no longer supports diodes, though you can find the diode implementation in the “diode” branch of the GitHub repository:

<https://github.com/sifferman/spice_replica/tree/diode>

After finishing diode support, I implemented the following model for solving MOSFETs:

<https://www.desmos.com/calculator/6dx8kxq8uo>

*Issue: I modified the MOS model so that all starting voltages converge on the correct value. However, this is only true assuming that the MOSFET drain and source are correctly labeled in the “.ckt” file. Otherwise, the voltage will incorrectly converge to 0. The correct behavior would be to have the tool swap the drain and source labels as current goes negative.*

*Issue: I did not have time to finish implementing parasitic capacitance. Therefore, the tool currently assumes negligible parasitic capacitances.*

# VII. Benchmark Circuits

This section contains all the waveforms for the provided “benchmark circuits”. I did not have time to compare with actual SPICE, but I expect that nearly all of them will match.

*Issue: I wasn’t sure what was meant by “branch current”, and there was no guidance in the lab parser. Therefore, those graphs are not displayed.*

## A. Clock Tree

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated Graphical user interface

Description automatically generatedGraphical user interface, chart

Description automatically generated with medium confidence Chart

Description automatically generated

Chart

Description automatically generated

## B. 3-Input NAND

Chart, histogram

Description automatically generatedChart

Description automatically generated

## C. RC Linear

Graphical user interface, chart, histogram

Description automatically generatedChart

Description automatically generatedChart

Description automatically generated

## D. RC Mesh

Graphical user interface

Description automatically generated Graphical user interface

Description automatically generated Graphical user interface

Description automatically generated Graphical user interface

Description automatically generated Graphical user interface

Description automatically generated Graphical user interface

Description automatically generated Graphical user interface

Description automatically generated Graphical user interface

Description automatically generated

## E. RLC Linear

Graphical user interface, chart, histogram

Description automatically generated Graphical user interface, chart

Description automatically generated Graphical user interface, chart, histogram

Description automatically generated Graphical user interface, chart

Description automatically generated

## F. Inverter

Graphical user interface

Description automatically generated Graphical user interface, chart

Description automatically generated

## G. Transmission Gate Mux

*Issue: This circuit is not able to be graphed due to divide-by-zero errors in the Newton-Raphson approximation. More investigation is required.*

1. [↑](#footnote-ref-1)