ELEC40006

Electronics design project

Circuit Simulation Report

Adam
Brandon Cann
Xin Wang

May 12, 2020

Contents

Abstract

This report describes the design and implementation of a program that is capable of performing a transient simulation by calculating the node voltages at each successive instant in time. This program parses the netlist file into a graph data structure, performs analysis using conductance matrices and outputs the results in a CSV format.

– How accurate is it?

– Comaparison to commercial software?

1 Overview of the report

This report is the distillation of multiple research documents relating to different components of the program.

Section 2 gives an abstract view of the design of the program, breaking the program down into 3 modules. Section 3 provides a summary of the testing methodologies and a comparison to both handwritten results and results of established circuit simulator software. Section 4 delves into the further work done and some potential ideas to build on. Section 5, the last section, summarises the report and discusses our overall experiences with the development of this project.

Talk about added functions and anything else.

2 Design of modules

The software package is spilt into 3 modules:

- Parser
- Analysis
- Interpreter
- Main

More in-depth information can be found under research papers of the respective topics. [1]

Object Orientated Programming approach has to be implemented.

2.1 Parse module

Format of an circuit description is [1]:

$$< \texttt{letter} >< \texttt{name} >< \texttt{node}i > ...[\texttt{modelname}][\texttt{parametervalues}]$$

From research, the best type of data structure to express a circuit is a **Graph Data Structure**. The graph data structure contains the following member variables:

- `letter`: Name of component
- `name`: Name of node
- `node1`: Name of node 1
- `node2`: Name of node 2
- `compval`: Value of the component e.g. 5 ohms or 3 volt

Aspects to consider regarding parse module:

- Identifying circuit elements
- Support for powers of ten

[1] Test scripts and their respective descriptions are found under "Tests Scripts" folder for reference.

  - When users are entering component values, it is important the program recognises common abbreviations for units
  - If an abbreviation is not recognised, it is ignored

Aspect to consider regarding storage component:

- Proper constructors and deconstructors are implemented

Block diagram depicting the breakdown of Parse Netlist module:

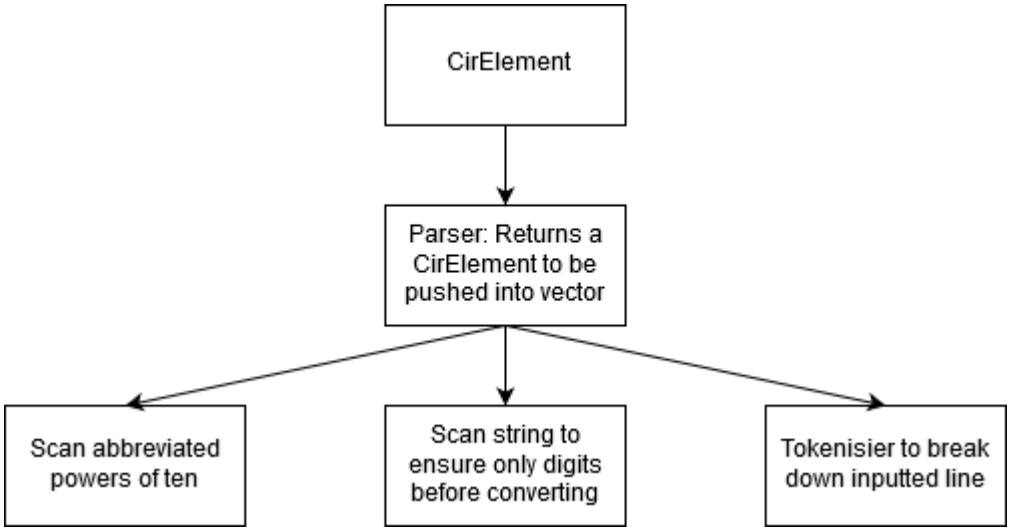

Figure 1: Netlist module breakdown

2.1.1 Pseudocode implementation:

Can be expanded to have a textfile containing the library of components supported which can be imported.

```
CirElement
{
  variables:

    letter: component name
    name: name of node
    node1: node this node is connected to
    node2: node this node is connected to
    value: float
```

```
    initial_val: float

  methods:

    parse(string: input)
    {
      Tokenise
      Put in values into respective variables
      Detect values, pass into custom_pow
      Detect if initial_val is entered:
        Pass into variable otherwise default 0
    }
    custom_pow(string: input)
    {
      Check if there are keywords e.g. k, m, M, G

      If not present, two scenario:
        Unknown letter present: extract digits
        Convert to float
        Empty string (End of recursion): return 0

      If present:
        Find position where keyword appears
        Take string before keyword and convert
        Multiply/divide the digit by keyword
        Recursion to cover case: 5M7k
    }
    tokeniser(string: input)
    {
      Call regex to tokenise the string
      Push each token into a vector
      Return vector
    }
    isdigit(string: input)
    {
      Iterate over string
      Take each character and into 'isdigit' test
      Return boolean
    }
}
```

2.2 Analysis module

3 Testing

3.1 *Data struct*

The script *Data struct test.sh*, when called, will compile *Data struct.cpp* and passes in input text file *Data struct input.txt*.

Pictures

This test is used to check the format of CirElement data structure functions as envisioned and that the methods associated with CirElement such as *custom pow* functions correctly.

4 Add-on

5 Conclusion

References

[1] Phyllis R. Nelson. *Introduction to spice source files*. DOI: https://www.cpp.edu/~prnelson/courses/ece220/220-spice-notes.pdf.