

Netlist Documentation

Adam, Brandon Cann, Xin Wang ¹

May 13, 2020

¹document compiled by Xin Wang

Contents

1	Introduction	2
2	Input (.cir) file	2
3	Netlist Element Format	2
3.1	Passive Elements in 'letter'	3
3.2	Independent Sources	3
3.3	Support for powers of ten	3
4	Network Diagram/Graphs	4
5	Implementation	4
5.1	struct CirElement	4
6	Miscellaneous	6

1 Introduction

According to the provided project briefing, a netlist describing the circuit will be provided in a file using a reduced SPICE format. It is required by the team to formulate an approach to read the file and store the information in a comprehensible data structure.

This document will research the format of the SPICE netlist and explore the different approaches in storing the information a certain way. This document is primarily used for planning the most effective way to approach the problem as well as allowing fellow teammates to engage productively on a equal knowledge footing.

2 Input (.cir) file

Source file for any version of SPICE has the following format [1].

```
TITLE
ELEMENT
.MODEL STATEMENTS
ANALYSIS COMMANDS
OUTPUT COMMANDS
.END
```

Important points to note:

- First line `TITLE` is used as a title on output files.
- File must end with command `.END` followed by a new line.
- Comment begins with `'*` which covers entire line.

3 Netlist Element Format

A circuit description is called a **netlist** which consists of statements defining a each circuit element.

Connections are described by naming nodes [Actually stored as numbers].

Format of an element description is:

```
< D > < description > < n1 > < n2 > [value][parameters]
```

- `< D >` : A character that is a unique identifier for each type of support circuit component
- `< description >` A string without space e.g. 5
- `[value]`: The value a circuit component takes e.g. 500 ohms

We have to consider the situation where the input syntax is correct but the circuit represented is wrong. To overcome this, we need a function to check if the circuit described is realistic.

3.1 Passive Elements in '**letter**'

Passive elements are composed of the following:

- Resistors: R

`R < description >< n1 >< n2 >< value >`

- Capacitor: C where we can specify the initial value of C for transient analysis.

`C < description >< n1 >< n2 >< value > [ic =< value >]`

- Inductor: L where we can specify the initial value of L for transient analysis.

`L < description >< n1 >< n2 >< value > [il =< value >]`

3.2 Independent Sources

- Voltage Sources

`V < description >< n1 >< n2 >< value >`

- Current Sources

`I < description >< n1 >< n2 >< value >`

The character after the letter must be a unique instance name and followed by the nodes associated with + and - respectively.

When users are entering component values, it is important that the software recognises common abbreviations for units.

3.3 Support for powers of ten

In order to support spice format, the following abbreviations for powers of ten must be recognised:

- f [femto]: 10^{-15}
- p [pico]: 10^{-12}
- n [nano]: 10^{-9}
- u [micro]: 10^{-6}

- m [milli]: 10^{-3}
- k [kilo]: 10^3
- M [mega]: 10^6
- G [giga]: 10^9
- T [tera]: 10^{12}

Any unrecognised characters are ignored.

4 Network Diagram/Graphs

Network Graphs show the relationships between a set of entries. Each entry is represented by a *Node* and the connections between nodes are represented through *Edges*.

There are four main types of network graphs:

- Undirected and Unweighted
- Directed and Unweighted
- Undirected and Weighted
- Directed and Weighted

A graph $G = (V, E)$ consists of: [2]

- a finite set $V = \{x_1, x_2, \dots, x_n\}$ whose elements are called nodes or vertices
- a subset E of the Cartesian product $V \times V$, the elements of which are called arcs.

The network graph defined will consists of two vector structures:

- Vector of nodes in the circuit
- Vector of branches in the circuit

The theoretical output is an ordered vector of branches and an ordered vector of nodes. **Position starts from 1 as node 0 indicates GND.**

5 Implementation

5.1 struct CirElement

```
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
}
}

```

6 Miscellaneous

There are more components to consider such as:

- Voltage controlled dependent sources
- Current controlled dependent sources
- Diodes
- BJT
- MOSFET
- Python implementation
- Efficient way to use recursion and not efficient to use matrix but more accurate.

References

- [1] Phyllis R. Nelson. *Introduction to spice source files*. DOI: <https://www.cpp.edu/~prnelson/courses/ece220/220-spice-notes.pdf>.
- [2] Archana Shankar, David Gilbert, and Michael Jampel. “Transient Analysis and Synthesis of Linear Circuits using Constraint Logic Programming”. In: (Oct. 1996).