الجمهورية الجزائرية الديمقراطية الشعبية
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
وزارة التعليم العالي و البحث العلمي
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

MOHAMED BOUDIAF UNIVERSITY OF M'SILA
FACULTY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

جامعة محمد بوضياف – المسيلة
كلية التكنولوجيا
قسم الإلكترونيك

الرقم : ...../ق.إ/2025 المسيلة في : 2025/11/24

شهادة إدارية

بعد الإطلاع على التقارير الايجابية الواردة من السادة الخبراء أعضاء لجنة دراسة المطبوعة الجامعية والاتية أسماؤهم:

- بن ذيب سارة أستاذ محاضر "أ" جامعة محمد البشير الابراهيمي– برج بوعريريج
- زواش طارق أستاذ محاضر "أ" جامعة محمد بوضياف – المسيلـــة
- حجاب مفدي أستاذ محاضر "أ" جامعة محمد بوضياف – المسيلـــة

صادق أعضاء اللجنـــة العلميـــة على قبول المطبوعة البيداغوجية مع إمكانية إتخاذها سندا في تدريس طلبة السنة الاولى ماستر ميكروالكترونيك ، في ميدان علوم و تكنولوجيا و أن تعتمد في أي تقييم المسار العلمي للأستاذ المعني **حرحوز أحلام** (أستاذ محاضر قسم "أ"– جامعة محمد بوضياف - المسيلة) تحت عنوان :

Simulation tools

رئيس اللجنة العلمية **رئيس القسم**

جامعة محمد بوضياف المسيلة – رئيس اللجنة العلمية – كلية التكنولوجيا

جامعة محمد بوضياف المسيلة – رئيس قسم الإلكترونيك – كلية التكنولوجيا

طباخ مصطفى

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION

AND SCIENTIFIC RESEARCH

Mohamed Boudiaf University - M'sila

M1-Microelectronic
Department of electronics

COURSE HANDOUT:

Simulation tools

Dr. HARHOUZ Ahlam

Course structuring and planning

Course: Simulation Tools

This module focuses on the essential tools and techniques for simulating electronic circuits, specifically using SPICE (Simulation Program with Integrated Circuit Emphasis). It is designed to equip students and professionals with the knowledge needed to analyze and optimize circuit designs before implementation.

Module Information

Faculty: Technologies

Department: Electronics

Target audience: Master 1, specialty Micro-Electronics

Course title: Simulation tools

Credit: 04

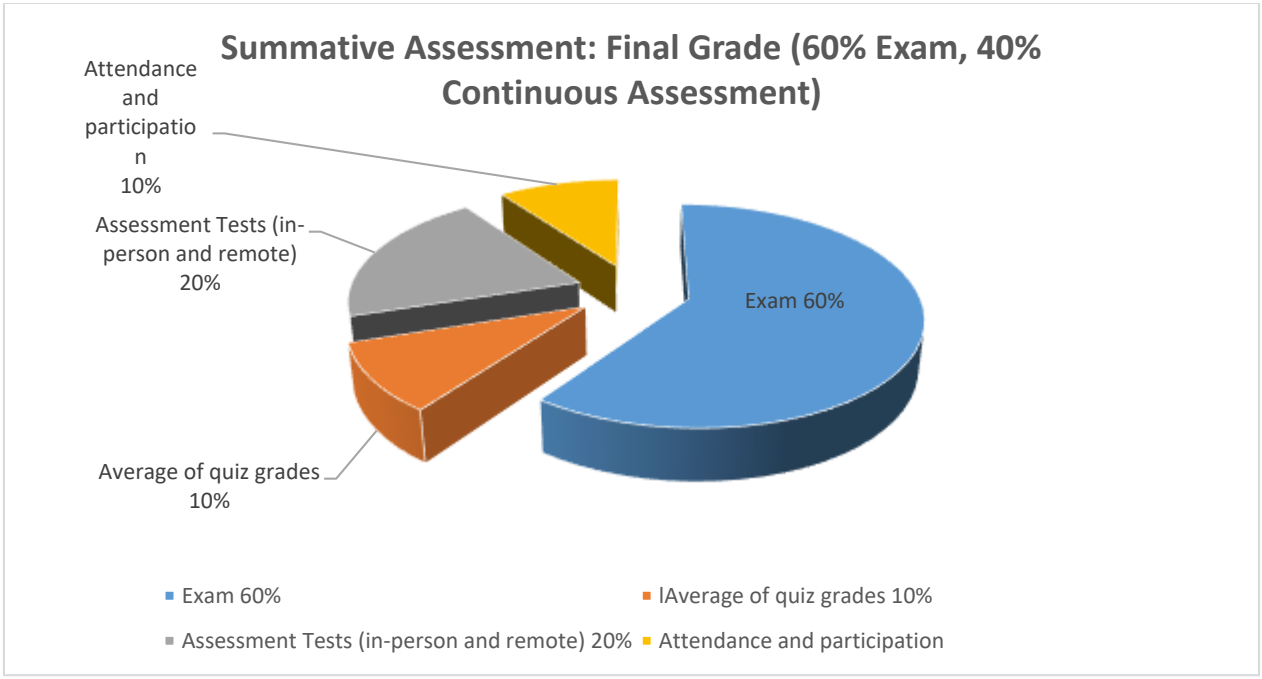
Coefficient: 02

Duration: 15 weeks Semester

Hourly Volume (15 weeks)

Additional Work in Consultation (15 weeks) 55h00

Assessment method: Continuous Assessment 40% Exam 60%



Teacher:

Dr. Ahlam HARHOUZ

Contact: by email at ahlam.harhouz@univ-msila.dz

Access link

https://moodle.univ-msila.dz/moodle/course/view.php?id=1723

Objectives:

This course titled "Simulation Tools" aims to help you:

- ✓ Become familiar with analog and digital simulators in electronics.
- ✓ Master the SPICE programming language.
- ✓ Describe analog circuits using the SPICE programming language.
- ✓ Understand and develop electronic schematics.
- ✓ Get acquainted with common analyses in electronics: transient, harmonic, and time-domain.
- ✓ Analyze circuits and implement numerical simulations.

Prerequisites:

To get the most out of this course you must have some acknowledgement:

1. Measurements & Instrumentation.
2. General device physics concepts
3. Basic fundamental *concepts of programming*
4. *Basic fundamental electrical/electronic component*
5. Signal theory and processing

Softwares Used:

Orcad 10.5 : PSCPICE

Course Contents:

1- Introduction

2- Chapter 1: Fundamentals of SPICE programming

3- Chapter 2: Description of an analog circuit and specification of components, (Declaring Models for Diodes, Bipolar Transistors, and MOS Transistors, Declaring Spice Models for Subcircuits)

4- Chapter 3: SPICE sources description. (Declaring Independent and Controlled Sources)

5- Chapter 4: Amplifiers specification.

6- Chapter 5: Analysis and Control Commands : (OP and DC Analysis, Transient and Fourier Analysis , Frequency and Noise Analysis, Temperature Analysis, Output Variables)

7- Exercises (DW)

REFERENCES

[1]. Lessons In Electric Circuits, Volume V– Reference By Tony R. Kuphaldt Fourth Edition, last update April 19, 2007

[2]. Lessons In Electric Circuits, Volume III Semiconductors By Tony R. Kuphaldt Fifth Edition, last update March 29, 2009

[3]. Cours de lectronique Remi Brendel Roger Bourquin 4 septembre 2008

[4]. SPICE, a guide to circuit simulation and analysis using PSpice , PAUL W.TUINENGA, MicroSim Corporation, PRENTICE HALL, Englewood Cliffs, New Jersy 07632

[5]. ANALYSE des circuits électriques et électroniques (simulation avec SPICE), presses international polytechnique, Simon Vatché chamlian and Chahé Nerguizian, 1999

[6]. Simon & Schuster Englewood Cliffs, New Jersey 07632, 1992 Sylvain Géronimi : Électronique analogique -Problèmes e t corrigés ; Université Paul Sabatier TOULOUSE III, 2010

Summary

Introduction

Generalities

Due to the increasing density and complexity of electronic circuits, simulation has become an essential step in the design cycle of circuits and components. It is crucial to verify in advance the electrical behavior of the circuit that will be sent to the manufacturer. Computer simulation serves as a powerful tool during the design phase or for the analysis of circuits and semiconductor components. It thus allows for a reduction in development costs by partially avoiding the creation of relatively expensive prototypes.

This module is primarily dedicated to the simulation of analog circuits. This technique could, of course, be applied to digital circuits (which are essentially composed of analog circuits), but the main limitation arises from the size of the circuits, as the methods presented here would provide an analysis that is too detailed for the studied circuit, making it prohibitively expensive in terms of resources (memory and computing time) to perform an analog analysis of a large digital system.

The most widely used analog simulation program today is "**SPICE**" (Simulation Program with Integrated Circuit Emphasis). This program was originally developed at the University of California, Berkeley, by Laurence Nagel in 1975, but research in the field of circuit simulation continues at many universities and companies.

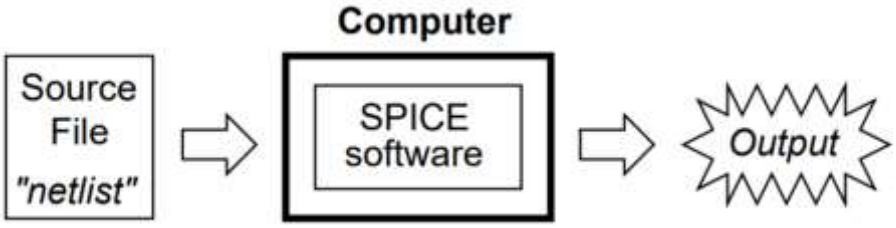
Free or commercial versions of **SPICE** or similar programs are currently available on a wide range of platforms, from personal computers to mainframes and workstations. Some notable versions include:

- SPICE2
- SPICE3 (Berkeley)
- PSpice (MicroSim/Orcad)

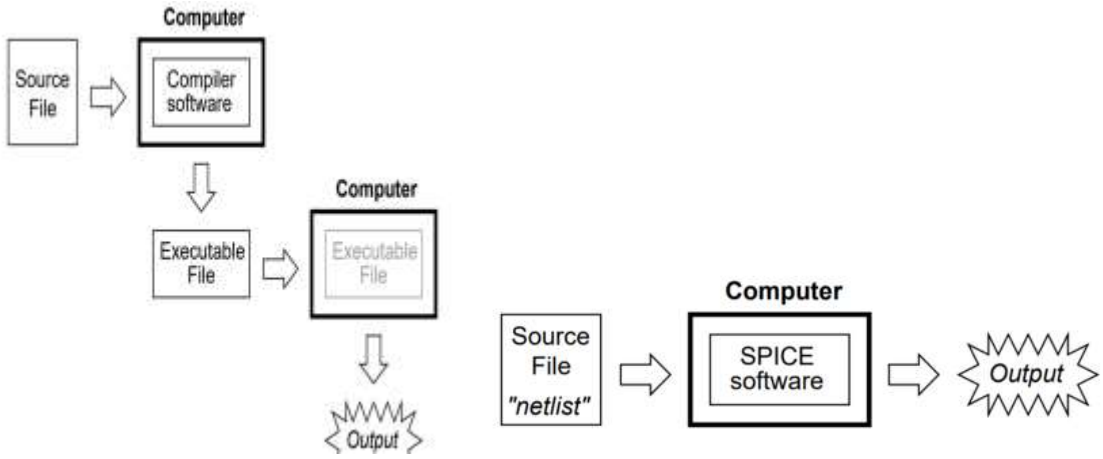
- Hspice (Meta Software)
- ISpice (Intusoft)
- Spectre (Cadence)
- Saber (Analogy)
- Smash (Dolphin Integration)

Fundamentals of SPICE programming

Programming a circuit simulation with SPICE is much like programming in any other computer language: you must type the commands as text in a file, save that file to the computer's hard drive, and then process the contents of that file with a program (compiler or interpreter) that understands such commands. In an interpreted computer language, the computer holds a special program called an interpreter that translates the program you wrote (the so-called source file) into the computer's own language, on the fly, as its being executed:

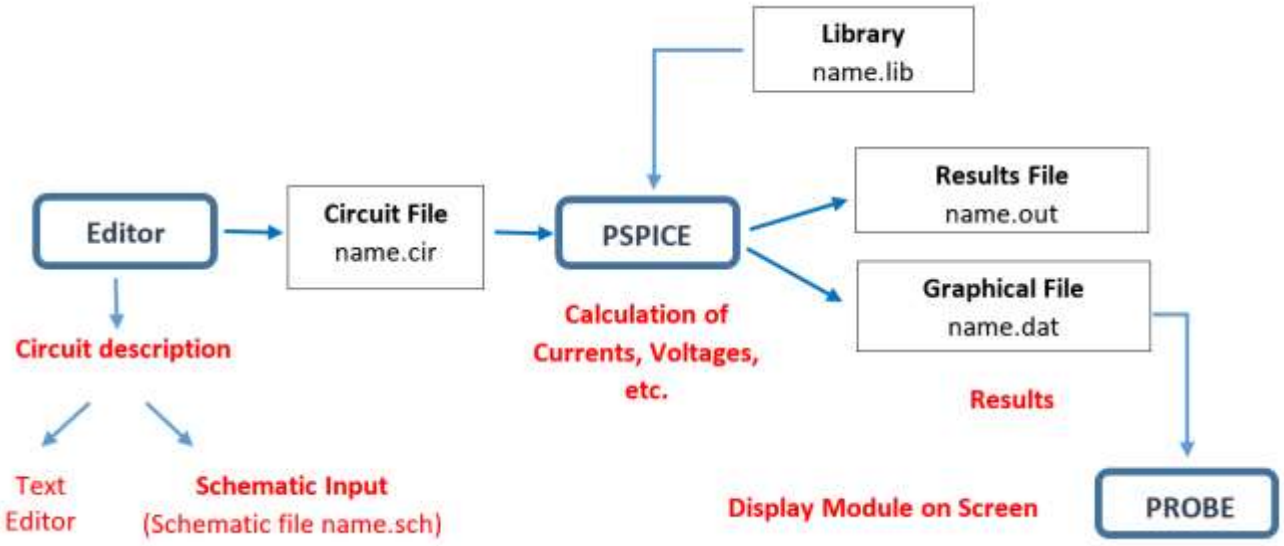


In a compiled computer language, the program you wrote is translated all at once into the computer's own language by a special program called a compiler. After the program you've written has been "compiled," the resulting executable file needs no further translation to be understood directly by the computer. It can now be "run" on a computer whether or not compiler software has been installed on that computer:



SPICE is an interpreted language. In order for a computer to be able to understand the SPICE instructions you type, it must have the SPICE program (interpreter) installed:

SPICE source files are commonly referred to as ”netlists,” although they are sometimes known as ”decks” with each line in the file being called a ”card.” Cute, don’t you think? Netlists are created by a person like yourself typing instructions line-by-line using a word processor or text editor. Text editors are much preferred over word processors for any type of computer programming, as they produce pure ASCII text with no special embedded codes for text highighting (like italic or boldface fonts), which are uninterpretable by interpreter and compiler software.



SPICE Overview

Step 1 (Input): The simulation of an electrical or electronic circuit always begins with the description of the circuit. This can be done using a text editor or a graphical schematic input program (preprocessor) that will convert the schematic into a text file. This file, called the "circuit file," also contains instructions regarding the analyses to be performed.

Step 2 (Calculation): To proceed with the simulation, the PSpice program must be executed, specifying the name of the circuit file: **name.cir**, where "name" is a user-defined filename and the extension **".cir"** is reserved.

Step 3 (Results): After the program has been executed, the results of the simulation are saved in the output text file name.out, and, if requested, the results are also saved in the binary file **name.dat** for further processing by the graphical post-processor Probe.

Chapter 1:

Fundamentals of SPICE programming

Introduction

Like most programming languages, the source file you create for SPICE must follow specific programming conventions. SPICE is a computer language in itself, albeit a simple one. As someone who has programmed in BASIC and C/C++ and has experience reading PASCAL and FORTRAN code, I find SPICE to be much simpler than any of these. Its complexity is comparable to a markup language like HTML, or perhaps even less.

Using SPICE to analyze a circuit follows a cyclical process. The cycle begins by creating the first draft of a netlist in a text editor. The next step is to run SPICE with this netlist and observe the results. Novice users will likely find their initial netlists contain minor syntax errors. However, as any experienced programmer knows, proficiency comes with practice. If a run produces error messages or clearly incorrect results, you must return to the text editor to modify the netlist. After making changes, run SPICE again to check the new results.

1. Netlist File

A netlist file is a standard ASCII file (with an extension like "*.CIR") that contains all the data characterizing the circuit to be analyzed: the components, nodes, parameters, analysis commands, models used, and desired outputs.

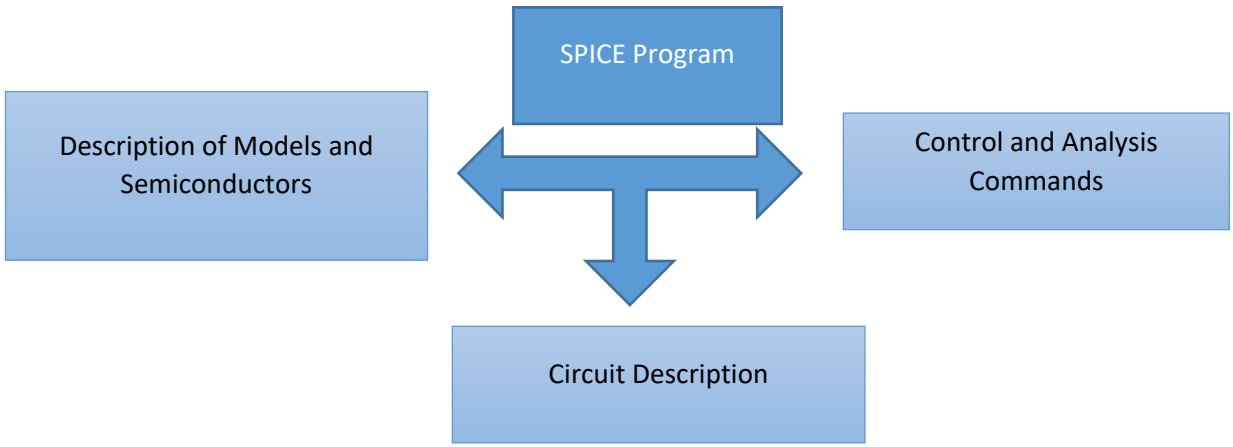


Fig. 1: components of a spice program

2. Structure of a Netlist

The recommended structure for a netlist file is as follows:

Netlist

- Title Declaration
- Models of Semiconductors
- Circuit Description (Power sources, signal sources, description of components, etc.)
- Analysis Request
- Output Request
- End

3. PICE Syntax

3.1. Typographic Conventions

- `<argument>` : mandatory argument
- `<argument>*` : the asterisk indicates that the argument must appear one or more times
- `[argument]` : optional argument
- `[argument]*` : zero, one, or multiple optional arguments.

3.2. Syntax

Data or instruction lines in a netlist file must be written in **UPPERCASE** or lowercase letters.

3.2.1 Data Lines

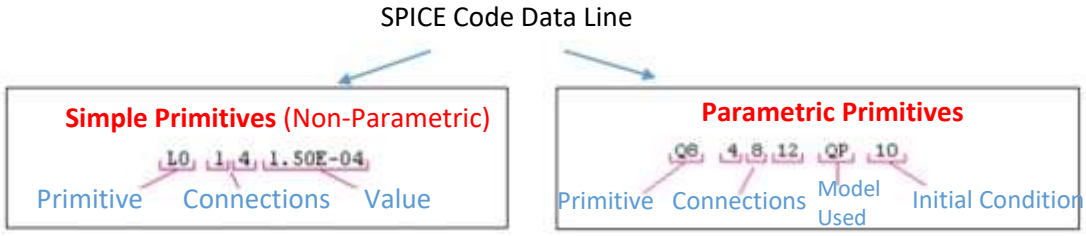


Fig. 2: SPICE Code Data Line

Data Line Structure in SPICE:

- The **element name** (the first character must be a letter).
- The **nodes** of the circuit to which the element is connected. Each node is identified by a positive integer, with node **0** reserved for ground.
- The **parameter values**; the numbers used can be integers, reals (in floating-point format xEy), or in a form using the suffixes listed below.

Table 1: Multiplicative Suffixes Recognized by SPICE

Suffixe	Meaning	value
F	**Femto**	10^{-15}
P	**Pico**	10^{-12}
N	**Nano**	10^{-9}
U	**Micro**	10^{-6}
M	**Milli**	10^{-3}
K	**Kilo**	10^{3}
MEG	**Mega**	10^{6}
G	**Giga**	10^{9}

Example

```
R1 2 3 120
```

Resistance Between Nodes 2 and 3 with a Value of 120Ω

< Here, `R1` is the name of the resistor, `2` and `3` are the nodes it connects, and `120` is the resistance value in ohms.>

```
R1 2 3 1.2e2
```

You can express the resistance value in several ways, such as: 120 120.00 1.2e2 1.2E2 0.12k

0.12kΩ. all of these representations are valid and denote a resistance of 120 ohms.

4. Analysis and Control Commands

All analysis commands are identified by a keyword that begins with a dot **(.)**. Each command follows its own specific syntax. The analyses can be classified into four main categories:

1. **DC Analysis**: Evaluates the circuit's behavior under direct current conditions.
2. **AC Analysis**: Analyzes the circuit's response to alternating current signals.
3. **Transient Analysis**: Examines the circuit's behavior over time in response to time-varying signals.
4. **Noise Analysis**: Assesses the noise performance of the circuit.

Operating Point Calculation (.OP): The .OP command allows for the output of detailed information about the operating point. The operating point is calculated even if there is no .OP instruction. Without the .OP command, the only information provided regarding the operating point is a list of node voltages. With the .OP command, the currents and power dissipated by all voltage sources are also reported.

The .OP command is used to calculate the operating point of the circuit. Its general form is quite simple:

Syntax:

```
.OP
```

This command does not require any additional parameters and is typically placed after the circuit description in the netlist.

It allows for the output of detailed information about the operating point. The operating point is calculated even if there is no .OP instruction.

- Without the .OP command, the only information provided regarding the operating point is a list of node voltages. With the .OP command, the currents and power dissipated by all voltage sources are also reported.

- Similarly, the bias state and small-signal (linearized) parameters for all controlled sources and semiconductor components are also provided.

- The .OP instruction only commands the output of normal operating point information. The .TRAN instruction commands the output of information regarding the operating point in transient analysis.

- The operating point is calculated after replacing inductors with short circuits and capacitors with open circuits.

DC Analysis (.DC): By nature, these analyses are time-independent and only concern the static behavior of the circuit (capacitors are treated as open circuits, and inductors as short circuits). These analyses take into account any static nonlinearities in the characteristics of the components.

The .DC command specifies a sweep of DC voltages. Its general form is as follows:

Syntax:

```
.DC <source> <start_voltage> <end_voltage> <increment>
```

- <source>: The voltage source to be swept.

- <start_voltage>: The starting voltage for the sweep.

- <end_voltage>: The ending voltage for the sweep.

- <increment>: The step size for the voltage sweep.

DC Analysis Syntax:

1. Linear Sweep:

```
.DC [LIN] <STOP1> [nested sweep specification]
```

2. Octave Sweep:

```
.DC [OCT] [DEC] <STOP1> [nested sweep specification]
```

3. List Specification:

```
.DC LIST * [nested sweep specification]
```

- <STOP1>: Indicates the stopping point for the sweep.
- [nested sweep specification]: Additional specifications for nested sweeps can be included.
- [LIN]: Indicates a linear sweep (default if not specified).
- [OCT]: Indicates an octave sweep.
- [DEC]: Indicates a decade sweep.
- LIST :uses a list of values. In this case, there are no start or end values. Instead, the variable will take on the successive values that follow the LIST keyword.

```
.DC LIST <value1> <value2> <value3> ... [nested sweep specification]
```

Here, each value provided will be used in the analysis.

Example

```
1    .DC VIN 0.25 5 0.25
2    .DC VDS 0 10 5 VGS 0 5 1
3    .DC VCE 0 10 0.25 IB 0 10u 1u
4
```

In the first statement, the source **VIN** initially has a value of 0.25 V and reaches a value of 5 V with a step increment of 0.25 V. The specification for the second source **SRC2** is optional, and its parameters (START2, STOP2, INC2) will produce a sweep for each value of the first

specified source, **SRC1**. This is a very practical option when trying to obtain the characteristics of transistors.

Small-Signal AC Analysis (.AC): The .AC command is used to calculate the frequency response of a circuit over a range of frequencies. The arguments LIN, OCT, or DEC are keywords that specify the type of sweep, and < nbr of Points > indicates the number of points used during the sweep.

General Forms:

Syntax:

```
.AC [LIN] [OCT] [DEC] < nbr of Points > <frequency_start> <frequency_end>
```

- LIN: Linear sweep. The frequency is swept linearly between the starting and ending frequencies. < nbr of Points > is the total number of points in the sweep.
- OCT: Octave sweep. The frequency is swept logarithmically by octaves. < nbr of Points > indicates the number of points per octave.
- DEC: Decade sweep. The frequency is swept logarithmically by decades. < nbr of Points> indicates the number of points per decade.

Transient Analysis (.TRAN): The .TRAN command allows for the execution of a transient analysis on the circuit. This analysis calculates the circuit's behavior over time, starting from TIME = 0 to the value of <final_time>. The transient analysis uses an internal time increment that is adjusted during the analysis.

General Forms:

Syntax:

```
.TRAN [/OP] <edit interval> <final_time> [<delay before output> + [max increment]] [SKIPBP] [UIC]
```

- /OP: Optionally specifies to perform an operating point analysis before the transient analysis.
- <edit interval> : Time interval for outputting results.
- <final_time>: The end time for the analysis.
- [<delay before output>]: Optional delay before output begins.
- [max increment]: Optional maximum time increment for the analysis.
- SKIPBP: If included, skips the calculation of the operating point.
- UIC: If included, uses initial conditions for reactive elements (capacitors and inductors) without calculating the operating point.

Example

```
.TRAN 1ns 100ns
.TRAN/OP 1ns 100ns 20ns SKIPBP
.TRAN 1ns 100ns 0ns .1ns
```

Example

```
.OP
.DC Ve -1 +1 0.1
.AC dec 21 1 1MEG
.TRAN 50u 5m
.STEP param Cvar LIST 100n, 220n, 470n, 680n, 1000n

```

SPICE provides a number of commands that allow you to set the initial state of components, define the characteristics of nonlinear components, structure the circuit file, specify the nature and form of variables to be output, and more.

Example

```
.OP
.PROBE
.PARAM vmax=10
.IC v(3)=0.1
.MODEL ca3086 NPN BF=175 VAF=50 CJE=5p CJC=3p
.END
```

5. Example of an Introduction

To illustrate the use of PSpice, let's consider the simple RC circuit represented in the following figure:

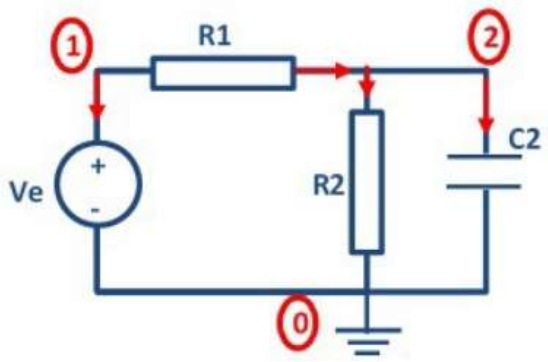


Fig. 2: Simple Circuit Showing Conventional Current Flow

This circuit consists of two resistors and a capacitor connected in series, which serves as a fundamental example for analyzing transient response and frequency behavior. Through this example, we will demonstrate how to set up the circuit in PSpice, perform simulations, and interpret the results to understand the circuit's performance.

In this simple circuit diagram, resistors (R1, R2) and a capacitor (C1) are connected in series to a voltage source (Ve). The direction of conventional current flow is indicated by arrows, showing that current flows from the positive terminal of the voltage source through the resistor, then to the second resistor and capacitor, and finally back to the negative terminal of the voltage source.

Program

```
Simple Pont RC
* Fichier rc.cir
R1 1 2 1K
R2 2 0 1K
C2 2 0 100n
```

```
* Source de tension
Ve 1 0 DC 1
+ AC 1
+ SIN(0 100m 10k)
.DC Ve -10 10 .02
.AC dec 10 1 10meg
.TRAN 1u 300u 0 1u
.Probe
.END
```

- The circuit file consists of lines describing components that always begin with an alphabetical character, and command lines that always start with a dot **(.)**.
- Each line constitutes a complete instruction; the first line is a title line, which can contain any text but is mandatory (if the file starts with the first instruction, it will be ignored). The last command line, **.END**, is also mandatory and marks the end of the circuit.
- Aside from the first and last lines, whose positions are dictated by syntax, other instruction lines can be placed in any order, and SPICE does not differentiate between lowercase and **uppercase** letters.
- If an instruction is too long, it can be continued on one or more consecutive lines; the first character of these continuation lines must be the plus sign **+**.
- The fifth line, which starts with an asterisk *****, is a comment line: any line starting with ***** is ignored by the command interpreter.

Chapter 2: Description of an analog circuit and specification of components

1. Instructions for describing components

An electrical circuit is made up of components placed between the nodes of a network. To describe its structure, it is sufficient to define the content of each branch. In PSPICE, each node has a number (which can also be a name) defined by the user or automatically assigned by the schematic entry software. One of these nodes is the ground "**GND**," which must have the number **0**.

The description file (NETLIST) reserves a line for each branch* this line begins with a letter characteristic of the component, **R** for a resistor, **C** for a capacitor, etc., followed by a number (or letters) indicating the component number.

Table 1 : Main electronic components

Category	Letter	Electrodes	Explanation
Passive Components	**R** **C** **L**	**N+ N-** **N+ N-** **N+ N-**	Resistance between N+ N- Capacitor between N+ N- Inductance between N+ N-
Independent Sources	**V** **I**	**N+ N-** **N+ N-**	Voltage source between N+ N- Current source between N+ N-
Controlled Sources	**E** **G** **H** **F**	**several cases**	Voltage Controlled by Voltage Current Controlled by Voltage Voltage Controlled by Current Current Controlled by Current
Semiconductors	**D** **Q** **M** **J**	**NA NC** **NC NB NE** **ND NG NS NB** **ND NG NS**	Junction Diode Bipolar Transistor MOSFET Transistor JFET Transistor

Depending on the nodes to which the component is connected (for a component with more than 2 terminals, several nodes may appear in a specific order that depends on the component). The table above specifies this order in each case. The line ends with the value of the component or its precise designation if it is an active component.

2. Specification of passive components

2.1. Resistors

The general format for describing a resistor is:

Syntax

```
R<name>  <N1 node>  <N2 node>  <Value>  [resistor specifications]*
```

- **R<name>**: The name of the resistor must start with the letter R.
- **N1** and **N2** are the terminals of the resistor.
- **<Value>:** is the value of the resistor (in Ohms).
- **[resistor specifications]***: **TC1** and **TC2** are the temperature coefficients. By default, **TC1** and **TC2** are equal to zero. If they are different from zero, then the value of the resistor is given by:

$$\boldsymbol{resistor\ value} = Value\,[1 + TC1(T - Tnom) + TC2(T - Tnom)^2]$$

Where:

- **TC1** is the linear temperature coefficient
- **TC2** is the quadratic temperature coefficient
- **Tnom** is the nominal temperature = 27°C by default.

Example

A resistor named R1 of 1KΩ placed between nodes 3 and 5

```
R1 3 5 1K
RCT 6 0 1.2KΩ TC=0.003, 0.01
```

For the value of the component, it is not necessary to specify the unit, as it is known by the program as soon as the type of component is specified. Therefore, the unit Ω is accepted but ignored."

2.2. Capacitors

The general format for describing a linear capacitor is:

Syntax

C<name> <N1 positive nodes> <N2 negative nodes> <Value> [+capacitor specifications]*

Where:

- **C<name>**: The name of the capacitor must start with the letter C
- **N1** and **N2** are the positive and negative nodes of the capacitor, respectively
- **<Value>**: The value indicates the capacitance value
- **[capacitor specifications]***: [**IC** = initial_voltage]: The initial condition for transient analysis is assigned using the command **IC** = initial_voltage on the capacitor.

Example

A capacitor named C1 of 22 nF placed between nodes 13 and 14

```
C1 13 14 22N
```

For the value of the component, it is not necessary to specify the unit, as it is known by the program once the type of component is specified. A letter, F or f for Farad, etc., is accepted but ignored.

2.3. Inductors

The general format for describing a linear inductance is:

Syntax

```
L<name>  <N1 positive nodes>  <N2 negative nodes>  <Value>

+ [inductance specifications]*
```

Where:

- **L<name>**: The name of the inductance must start with the letter L
- **N+** and **N-** are the positive and negative nodes of the inductance, respectively
- **<Value>**: The value indicates the inductance value
- **[inductance specifications]***: [**IC** = initial_current]: The initial condition for transient analysis is assigned using the command **IC** = initial_current on the inductance.

Example

An inductance named L10 of 2.2 mH placed between nodes 4 and 7.

```
1   L10 4 7 2.2M
```

Note that **M** means milli and not mega. Uppercase and lowercase are equivalent.

2.4. Mutual inductance

The general format for describing mutual inductance is:

Syntax

```
k<name> L<name1>  L<name2>  <Value-k>
```

Where:

- **K<name>**: The name of the mutual inductance must start with the letter **K**
- **L<name1>** and **L<name2>** are the names of the input and output inductors, respectively
- **<Value-k>**: is the coupling coefficient between **L1** and **L2**.

Example

```
K1 L12 L20 0.6
```

K1 L12 L20 0.6 couples the inductors L12 and L20 with a coupling coefficient of 0.6 (for example, to make a transformer). The inductors L12 and L20 must be defined elsewhere, using

```
Ln  n1  n2  val.
```

2.5. Nonlinear capacitors and inductors

The general format for describing a non-linear capacitor or inductance is:

Syntax

```
L<name>  <N+>  <N->  POLY  <L0>  <L1>  <L2> [inductance specifications]*

C<name>  <N+>  <N->  POLY  <C0>  <C1>  <C2> [capacitor specifications]*
```

C0, **C1**, **C2**, ..., **L0**, **L1**, **L2**, ... are the coefficients of a polynomial describing the value of the element. Capacitance is expressed as a function of the voltage across the capacitor, while inductance is typically a function of the current flowing through the coil. To determine the values of capacitance and inductance for the non-linear elements calculated in the **POLY** expression, we use respectively...

3. Component models: Instruction .model

For semiconductor components (diodes, BJTs, FETs, MOS, etc.) or, more generally, any components that cannot be described by a simple numerical value, referring to a model is not

only useful but also mandatory. Instead of a numerical value, the component is described by a <name> of the model, which refers to the .MODEL command that bears the same <name>.

General Form

Syntax:

```
.MODEL <name>  <type>  ([<parameter name>=<value> [tolerance specification]]*)
```

Each component has a sometimes significant number of model parameters described in the SPICE documentation, which should be consulted to understand their meanings.

Example

In the example below, component D1 is a diode connected between nodes 1 and 2, described by the model "diode." This name is arbitrary and can be made up of any alphanumeric string that does not contain a separator. When the command interpreter encounters this instruction, it searches for the **.MODEL** command with the same reference. This instruction specifies the <type> of component described by a reserved keyword, the list of which is given in Table 2. Here, we see that the same component can have different <types>, which explains the need to specify it in the model. It is also noted that several components can reference the same model (diodes **D1** and **D2**), and that the description of passive components can also be supplemented by a model.

```
Vin 1 0 SIN(0 10V 50Hz)
 D1 1 2 diode
 D2 2 1 diode
 R 2 0 modR 1
 C 2 0 modC 1
.MODEL diode D (IS=1e-12)
.MODEL modR RES (R=1k Tc1=.02)
.MODEL modC CAP (C=1000u)
```

Other Examples

```
.MODEL DNOM D (IS=1E-9)
.MODEL QDRIV NPN (IS=1E-7 BF=30)
.MODEL MLOAD NMOS (LEVEL=1 VTO=.7 CJ=.02PF)
.MODEL DLOAD D (IS=1E-9 DEV .5% LOT 10%)
```

Table 2 : Types of SPICE Models

Type	Component	Nature
CAP	**C**	Capacitor
IND	**L**	Inductance
RES	**R**	Resistance
D	**D**	Junction Diode
NPN	**Q**	Bipolar Transistor NPN
PNP	**Q**	Bipolar Transistor PNP
PNP	**Q**	Lateral Bipolar Transistor PNP
NMOS	**M**	N-Channel MOSFET Transistor
PMOS	**M**	P-Channel MOSFET Transistor
GASFET	**B**	N-Channel MOSFET AsGaTransistor
NJF	**J**	N-Channel Junction FET Transistor
PJF	**J**	N-Channel Junction FET Transistor

4. Specification of semiconductor components

4.1. Diodes

The general format for describing a diode is:

Syntax

```
D<name> <NA> <NC> <model> <AREA>
```

where:

- **NA**: node connected to the anode
- **NC**: node connected to the cathode
- **AREA**: a multiplicative factor representing the number of identical diodes connected in parallel; the default value of the **AREA** parameter is **1**.

Model format: Syntax

```
.MODEL <model name>  D [model parameters]
```

Example

```
1    D2  4  2  DN4148
```

The data line describes a diode (characteristic letter D) named D1, of model type DN4148, placed between nodes 4 and 2.

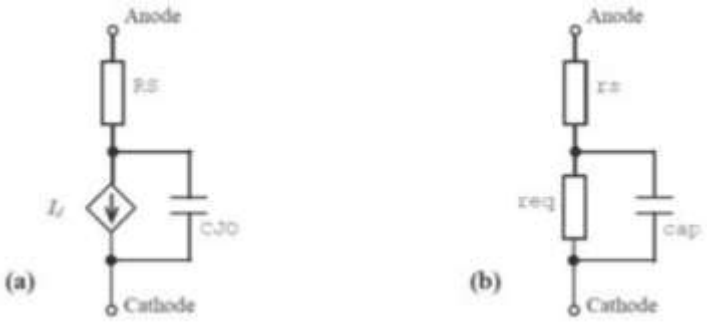


Fig. 1: Diode models used by Pspice: (a) nonlinear model for .DC and .TRAN analyses. (b) linear model for .AC analysis.

Table 3: List of main parameters of a diode in Spice language.

Parameter	Definition	PSpice Unit	Default Value
IS	Saturation current	Amperes	10^-14
N	Emission coefficient		1
RS	Parasitic resistance	Ohms	0
Cj0	Junction capacitance at zero bias	Farads	0
TT	Transit time	Seconds	0
BV	Reverse avalanche voltage	Volts	∞
IBV	Reverse avalanche current	Amperes	10^-3
M	Grading coefficient		0.5
Eg	Activation energy depending on the type of diode (silicon diode, Schottky diode, etc.)	Electronvolts	1.1

Additional Information

To determine the behavior of a circuit containing a diode, PSpice always starts by calculating the static bias point of the circuit using the nonlinear diagram shown in Figure 4 (a). It then determines the small-signal linear parameters of the equivalent circuit in Figure (b). This linear equivalent circuit is only used for AC-type analyses. For other nonlinear analyses (DC and transient), the diagram from Figure 4 (a) is used (note that capacitance is only relevant for transient analyses).

Example: Circuit file for determining the current-voltage characteristic of a diode.

```
* Caracteristique de diode *
*
I 0 1 10m ac=1
D 1 0 diode
.model diode D(Is=1e-16 cjo=2p)
.dc I 0 30m 50u
.op
.probe
.end

```

4.2. Bipolar Junction Transistors (BJT)

The general format for describing a bipolar transistor is:

Syntax

```
Q<name> <NC> <NB> <NE> <model> <AREA>
```

where:

- **NC**: node connected to the collector
- **NB**: node connected to the base
- **NE**: node connected to the emitter
- **AREA**: a multiplicative factor representing the number of identical transistors connected in parallel; the default value for the AREA parameter is 1.

Model form:

Syntax

```
.MODEL <model name> NPN [model parameters]

.MODEL <model name> PNP [model parameters]
```

Example

```
.MODEL NPN_MODEL NPN (IS=1E-14 BF=100)
```

- NPN_MODEL: Model name
- NPN: Specifies it's an NPN transistor
- **IS**=1E-14: Saturation current
- **BF**=100: Forward current gain

```
.MODEL PNP_MODEL PNP (IS=1E-14 BF=80)
```

- **PNP_MODEL**: Model name
- **PNP:** Specifies it's a PNP transistor
- **IS**=1E-14: Saturation current

- **BF**=80: Forward current gain

Other Examples

```
*NPN Transistor
V1 1 0 DC 10 *DC Voltage Source
R1 1 2 1k *Resistor (1k ohm)
Q1 2 3 0 NPN_MODEL AREA=1 *NPN Transistor
.MODEL NPN_MODEL NPN (IS=1E-14 BF=100) *NPN Model Definition
.dc V1 0 10 1 *DC Sweep from 0 to 10V
.end
```

```
* AC Analysis of NPN Transistor
Vcc 1 0 DC 15    *Supply Voltage
R1 1 2 1k        *Base Resistor
C1 2 0 1u            *Coupling Capacitor
Q1 2 3 0 NPN_MODEL AREA=1      *NPN Transistor
.MODEL NPN_MODEL NPN (IS=1E-14 BF=150)     *NPN Model
Definition
.ac dec 20 1k 100k         *AC Analysis from 1kHz to 100kHz
.end
```

```
* Cascaded NPN and PNP Transistor Circuit
Vcc 1 0 DC 15 *Power Supply
R1 1 2 1k   *Base Resistor for NPN
R2 2 3 1k   *Collector Resistor for NPN
Q1 2 4 0 NPN_MODEL AREA=1          *NPN Transistor
```

```
Q2  4  5  0  PNP_MODEL  AREA=1          *PNP Transistor
.MODEL  NPN_MODEL  NPN  (IS=1E-14  BF=100)    *NPN Model Definition
.MODEL  PNP_MODEL  PNP  (IS=1E-14  BF=80)    *PNP Model Definition
.dc  Vcc  0  15  1      *DC Sweep
.end
```

Table 4: List of main parameters of a BJT transistor in Spice language.

Parameter	Description	Unit	Default Value
IS	Saturation current	Amperes (A)	1×10^{-14}
BF	Forward current gain	Dimensionless	100
BR	Reverse current gain	Dimensionless	0.1
VAF	Early voltage	Volts (V)	100
IKF	Maximum forward current	Amperes (A)	0.5
XTB	Temperature exponent for beta	Dimensionless	1.5
NE	Emission coefficient	Dimensionless	1
RE	Emitter resistance	Ohms (Ω)	0.5
RC	Collector resistance	Ohms (Ω)	0.5
RB	Base resistance	Ohms (Ω)	10
CJE	Base-emitter junction capacitance	Farads (F)	1×10^{-12}
CJC	Base-collector junction capacitance	Farads (F)	1×10^{-12}
VJE	Junction potential of the base-emitter	Volts (V)	0.75
MJE	Junction capacitance grading coefficient	Dimensionless	0.5
TF	Forward transit time	Seconds (s)	1×10^{-9}
TR	Reverse transit time	Seconds (s)	1×10^{-9}

4.3. Junction Field Effect Transistors (JFET)

The general format for describing a field-effect transistor (FET) is:

Syntax

```
J<name> <ND> <NG> <NS> <model> <AREA>
```

where:

- **ND**: node connected to the drain
- **NG**: node connected to the gate
- **NS**: node connected to the source

Model form:

Syntax

```
.MODEL <model name> J [model parameters]
```

Example

```
J2 1 3 2 J105
```

The data line designates a field-effect transistor (characteristic letter J) named J2, with the Drain, Gate, and Source connected to nodes 1, 3, and 2, respectively, and this transistor is a J105.

Other Examples

```
* Simple JFET Example
V1 1 0 DC 10 *DC Voltage Source
R1 1 2 1k *Resistor (1k ohm)
J1 2 3 0 JFET_MODEL AREA=1 *JFET Transistor
.MODEL JFET_MODEL J (ID=1m VTH=-2 K=5) *JFET Model Definition
.dc V1 0 10 1 * DC Sweep from 0 to 10V
.end
```

```
* AC Analysis of JFET
```

```
Vcc 1 0 DC 15 *Supply Voltage
R2 1 2 1k *Gate Resistor
C1 2 0 1u *Coupling Capacitor
J2 2 3 0 JFET_MODEL AREA=1 *JFET Transistor
.MODEL JFET_MODEL J (ID=1m VTH=-2 K=10) *JFET Model Definition
.ac dec 20 1k 100k *AC Analysis from 1kHz to 100kHz
.end
```

```
*Cascaded JFET Circuit
Vcc 1 0 DC 12 *Power Supply
R1 1 2 1k *Resistor
C1 2 0 1u *Coupling Capacitor
J1 2 3 0 JFET_MODEL AREA=1 *First JFET
J2 3 4 0 JFET_MODEL AREA=1 *Second JFET
.MODEL JFET_MODEL J (ID=1m VTH=-2 K=5) *JFET Model Definition
.dc Vcc 0 12 1 * DC Sweep
.end
*JFET with Temperature Effects
Vgs 1 0 DC 3 *Gate-Source Voltage
Rload 2 0 1k *Load Resistor
J3 2 1 0 JFET_TEMP_MODEL AREA=1 *JFET Transistor
.MODEL JFET_TEMP_MODEL J (ID=1m VTH=-2 K=15) *JFET Model with
Temperature
.temp 25  *Set temperature to 25°C
.dc Vgs 0 5 0.5 *DC Sweep for Vgs
```

```
.end
```

Table 5: List of main parameters of a JFET in Spice language.

Parameter	Description	Unit	Default Value
ID	Drain current at zero gate-source voltage	Amperes (A)	1×10^{-3}
VGS	Gate-source cutoff voltage (threshold voltage)	Volts (V)	-2.0
VP	Pinch-off voltage	Volts (V)	-3.0
K	Transconductance parameter	mA/V^2	5.0
RD	Drain resistance	Ohms (Ω)	100
RS	Source resistance	Ohms (Ω)	10
CGS	Gate-source capacitance	Farads (F)	1×10^{-12}
CGD	Gate-drain capacitance	Farads (F)	1×10^{-12}
M	Capacitance grading coefficient	Dimensionless	0.5
XJ	Junction depth	Meters (m)	1×10^{-7}

4.4. MOSFET Transistors

The general format for describing a MOSFET (Metal-Oxide-Semiconductor Field-Effect Transistor) is:

Syntax

```
M<name>  <ND>  <NG>  <NS>  <NB> <model> <AREA>
```

where:

- **ND**: node connected to the drain
- **NG**: node connected to the gate
- **NS**: node connected to the source
- **NB**: node connected to the substrate (or body)

Model form:

Syntax

.MODEL <model name> M [model parameters]

Example

```
M7 3 8 0 9 Mnmos
```

The data line designates a MOSFET (characteristic letter M) named M7, with the Drain, Gate, Source, and Substrate connected to nodes 3, 8, 0, and 9, respectively. This transistor is of type N.

Other Examples

```
* N-Channel MOSFET Example
Vds 1 0 DC 10 * Drain-Source Voltage
Vgs 2 0 DC 5 * Gate-Source Voltage
Rload 1 0 1k * Load Resistor
M1 1 2 0 0 NMOS_MODEL * N-Channel MOSFET
.MODEL NMOS_MODEL NMOS (VTH=2 K=100) * NMOS Model Definition
.dc Vgs 0 5 1 * DC Sweep for Vgs
.end
```

```
* MOSFET Inverter Example
Vdd 1 0 DC 10 * Supply Voltage
Vin 2 0 PULSE(0 10 0 1n 1n 10u 20u) * Input Pulse
M1 1 2 0 0 NMOS_MODEL * N-Channel MOSFET
M2 1 2 0 0 PMOS_MODEL * P-Channel MOSFET
.MODEL NMOS_MODEL NMOS (VTH=2 K=100) * N-Channel Model
.MODEL PMOS_MODEL PMOS (VTH=-2 K=100) * P-Channel Model
```

```
.tran 0.1u 100u * Transient Analysis
.end
* H-Bridge Example
V1 1 0 DC 12 * Supply Voltage
V2 2 0 DC 5 * Control Voltage for M1
V3 3 0 DC 5 * Control Voltage for M3
Rload 4 0 10 * Load Resistor
M1 1 2 0 0 NMOS_MODEL * Upper Left
M2 2 4 0 0 PMOS_MODEL * Lower Left
M3 3 0 0 0 NMOS_MODEL * Upper Right
M4 4 0 0 0 PMOS_MODEL * Lower Right
.MODEL NMOS_MODEL NMOS (VTH=2 K=100) * N-Channel Model
.MODEL PMOS_MODEL PMOS (VTH=-2 K=100) * P-Channel Model
.dc V2 0 5 1 * DC Sweep for control voltage
.dc V3 0 5 1 * DC Sweep for control voltage
.end
```

Table 6: List of main parameters of a MOSFET in Spice language.

Parameter	Description	Unit	Value
VTH	Threshold voltage	Volts (V)	2.0
K	Transconductance parameter	mA/V^2	100
KP	Process transconductance parameter (for enhancement mode)	mA/V^2	100
VDS	Drain-source voltage	Volts (V)	-
VGS	Gate-source voltage	Volts (V)	-
GAMMA	Body effect coefficient	V½	0.4
PHI	Surface potential	Volts (V)	0.6
RD	Drain resistance	Ohms (Ω)	0.1
RS	Source resistance	Ohms (Ω)	0.1
CGS	Gate-source capacitance	Farads (F)	1×10^{-12}
CGD	Gate-drain capacitance	Farads (F)	1×10^{-12}
M	Capacitance grading coefficient	Dimensionless	0.5
L	Channel length	Meters (m)	1×10^{-6}
W	Channel width	Meters (m)	1×10^{-6}

5. Subcircuits

The concept of a **subcircuit** is analogous to the concept of a subroutine in programming languages. It allows for structuring a circuit file into blocks that can be reused within the same circuit file or compiled into a library of subcircuits for use in another circuit file.

A subcircuit block is enclosed by the commands **.SUBCKT** and **.ENDS** and is called like a component whose identifier starts with the letter **X**.

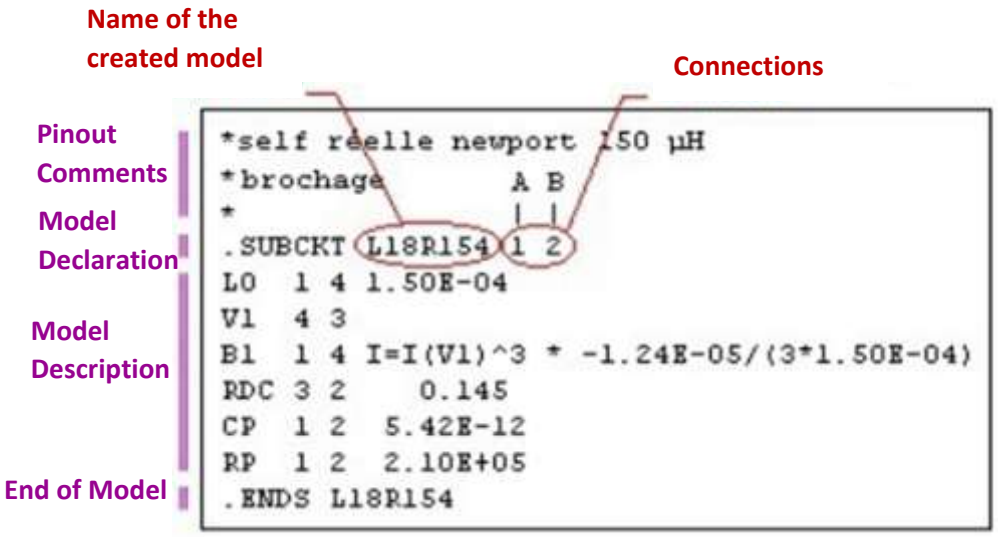
The general format for describing a resistor is:

Syntax

```
.SUBCKT <subcircuit name>  <node>... <subcircuit>
```

.ENDS [subcircuit name]

A .SUBCKT file allows for the creation of a macro-circuit, which is essentially a circuit resulting from the combination of several SPICE primitives. For all versions of SPICE, a .SUBCKT file is structured as follows:



Additional Information

- The **.SUBCKT** statement begins the definition of a subcircuit.
- The **.ENDS** statement indicates the end of this definition
- . All statements between **.SUBCKT** and **.ENDS** are part of the definition.
- Each time a subcircuit is called using an **X** statement, all the instructions in the definition replace the calling statement.
- **<subcircuit name>** is the name used by the X statement to reference the subcircuit. It must start with a letter.
- **<node>*** is a list of nodes. There must be as many nodes in the calling statement as in the definition.

- When the subcircuit is called, the actual nodes (those from the calling statement) replace the argument nodes (those from the definition statement).
- Subcircuit calls can be nested, meaning an X statement can appear between a **.SUBCKT** and a **.ENDS**. However, subcircuit definitions cannot be nested; a **.SUBCKT** statement cannot appear between a **.SUBCKT** and an **.ENDS**.
- Subcircuit definitions should only contain component description statements (statements without an initial "**.**"; no control or command statements), although the use of **.MODEL** statements is permitted.
- Models defined within a subcircuit definition are only accessible within the definition of that subcircuit. Conversely, if a **.MODEL** statement appears in the main circuit, that model is accessible in the main circuit and all subcircuits.
- The names of nodes, components, and models are local to the subcircuit in which they are defined. This means it is acceptable to use a name in a subcircuit that has already been used in the main circuit.

General form of the subcircuit call:

Syntax

```
X<name>  <node>*  <subcircuit name>
```

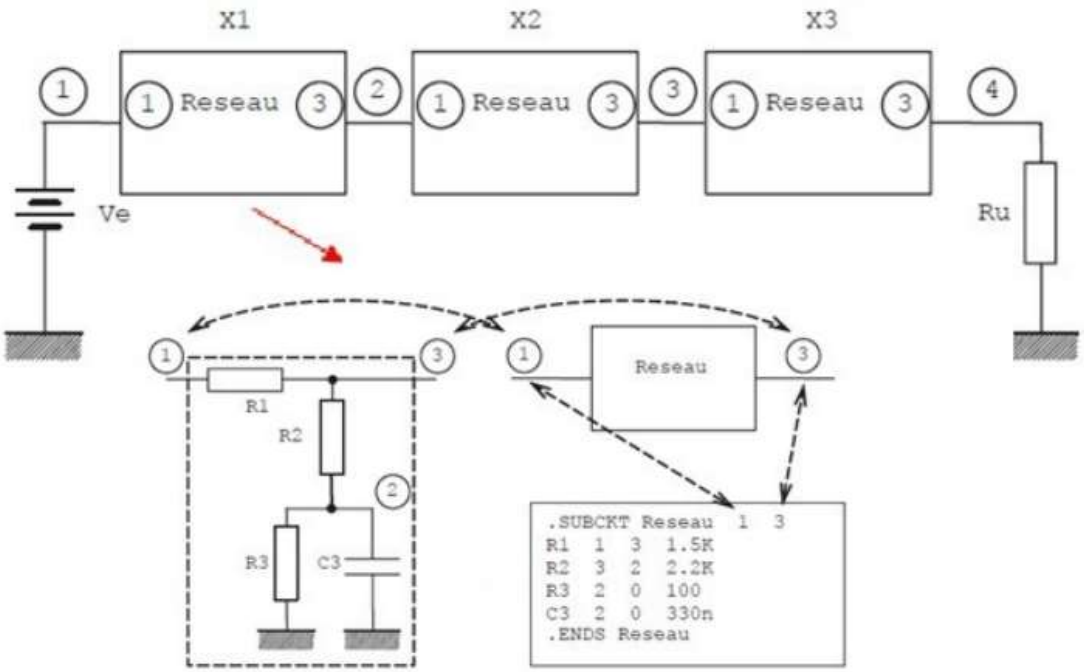


Fig. 2: Use of Subcircuits (Initial Circuit, Subcircuit, Subcircuit Text)

Figure 2 shows a circuit containing three identical cells. This cell can be described in a subcircuit called "Reseau" and used in three instances within the main circuit. Note that the "Reseau" subcircuit has three internal nodes, two of which (1 and 3) are accessible from the outside. Also, observe the structure of the main circuit and the call to the subcircuits using the X statement. PSpice-specific extensions also allow the passing of parameters into a subcircuit, as illustrated in Figure 2. This capability enables the creation of subcircuits without needing to know the values of the internal components a priori.

```
* Sous circuits *
.SUBCKT Reseau 1 3
 R1 3 1.5K
 R2 3 2 2.2K
 R3 2 0 100
 C3 2 0 330n
 .ENDS Reseau
 Ve 1 0 1V
 X1 1 2 Reseau
 X2 2 3 Reseau
 X3 3 4 Reseau
 Ru 4 0 120
.OP
.DC Ve 0 20 5
.Print DC I(X1.R1) I(X3.R2) I(Ru)
.END
```

Chapter 3:

Specification of

Sources

1. Introduction

Due to their presence in electrical circuits, it is necessary to declare voltage and current sources in circuit simulation programs. PSPICE provides two types of sources: independent sources (current or voltage) and controlled sources (current or voltage). In this chapter, we will explore how to specify these different sources.

2. Independent Source

I - Current source, V - Voltage source

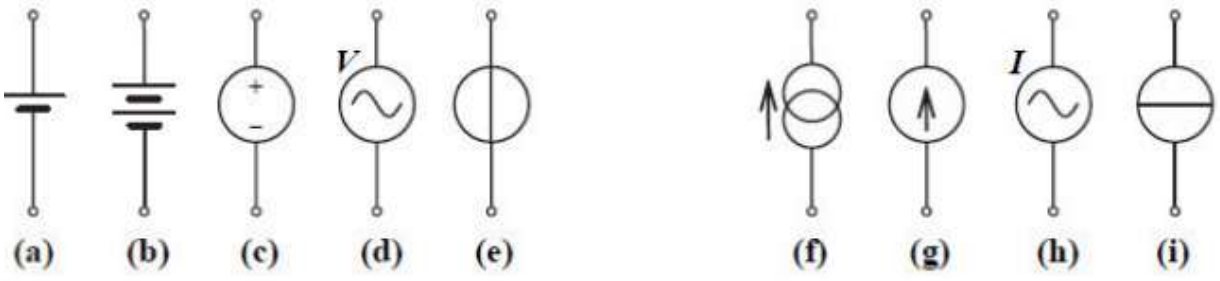


Fig. 1: Symbols for Independent Voltage Sources (a-e) and Current Sources (f-i)

2.1 DC and AC Sources

The general formats for describing a voltage or current source are as follows:

Syntax:

```
I< Name >     < N1 nœuds positifs > < N2 nœuds négatifs >
+[[DC] ][AC [phase]][spécifications de transitoire]
V< Name >     < N1 nœuds positifs > < N2 nœuds négatifs >
+[[DC] ] [AC [phase]] + [spécifications de transitoire]
```

Where:

- **N1** and **N2** are the positive and negative nodes of the source.
- **DC** value: the value of the DC component.
- **AC** amplitude phase: the amplitude and phase of the AC component.
- **<value>:** the default value is zero for DC, AC, or transient specifications.
- **<amplitude>**: the default value is 1 for the AC amplitude.
- **[phase]**: the default value is 0 for the AC phase.

  The phase of the AC specification is expressed in degrees.

If a transient mode is specified, it must be one of the following keywords:

EXP <parameters>: exponential waveform signal,

PULSE <parameters>: pulse or square waveform signal,

PWL <parameters>: piecewise linear waveform signal,

SFFM <parameters>: frequency-modulated sinusoidal signal,

SIN <parameters>: sinusoidal signal.

2.2 Time Specifications for Sinusoidal Waveforms (SIN)

The general formats for describing a sinusoidal voltage or current source are:

Syntax:

```
V< Name >    < N1 nœuds positifs > < N2 nœuds négatifs>
+[SIN (Voff Vampl freq td df phase)]
I< Name >    < N1 nœuds positifs > < N2 nœuds négatifs >
 +[SIN (Voff Vampl freq td df phase)]
```

Where:

- **Voff:** DC component value
- **Vampl:** Amplitude of the AC component

- **freq:** Frequency in Hz
- **td:** Signal delay (0 by default)
- **df:** Damping coefficient (0 by default)
- **phase:** Phase (0 by default)

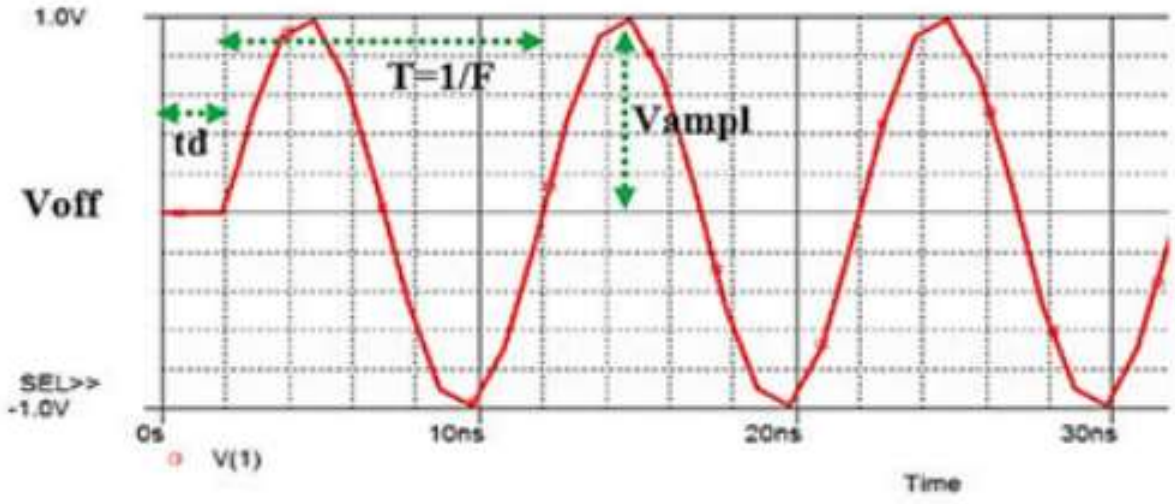


Fig. 2: Sinusoidal Waveforms

Variable frequency sinusoidal source

In several circuit analyses, the frequency response of the circuit is of interest. The frequency response indicates the amplitude and phase of the voltage at each node of the circuit, as well as the amplitude and phase of the current in each branch of the circuit as a function of frequency. To perform this type of analysis, it is necessary to specify a sinusoidal source (AC) using the **.AC** command to define the desired frequency sweep range (starting frequency, frequency increment, stopping frequency).

Syntax of a Sinusoidal (AC) Source:

```
V< Name >     < N1 nœuds positifs > < N2 nœuds négatifs >
 +[AC < ACAMP > < ACPHSE >]
I< Name >     < N1 nœuds positifs > < N2 nœuds négatifs >
+[AC < ACAMP > < ACPHSE >]
```

Where:

- **ACAMP**: Amplitude (when the value of ACAMP is not specified, the program assigns it a default value of 1)
- **ACPHSE**: Phase in degrees (0 by default)

2.3 Time Specifications for Frequency-Modulated Sinusoidal Waveforms (SFFM)

The general formats for describing a frequency-modulated sinusoidal voltage or current source are as follows:

Syntax:

```
V < Name >   < N1 nœuds positifs > < N2 nœuds négatifs>
+ [SFFM(Voff Vampl freq mod fs)]
I< Name >    < N1 nœuds positifs > < N2 nœuds négatifs>
 +[SFFM (Voff Vampl freq mod fs)]
```

Where:

- **Voff:** Offset value, average value
- **Vampl:** Amplitude
- **freq:** Carrier frequency
- **mod:** Modulation index (0 by default)
- **fm:** Modulation frequency

The SFFM signal is described by the following expression:

$$v(t) = Voff + Vampl \cdot [sin(2\pi \cdot freq \cdot t + mod \cdot sin(2\pi \cdot fm \cdot t))]$$

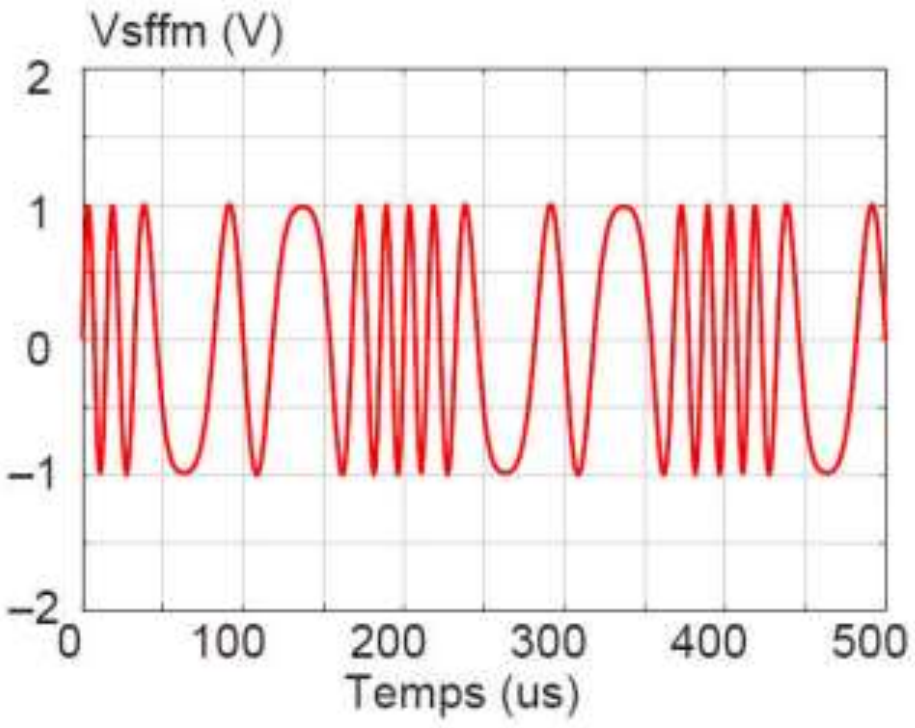


Fig. 3: Sinusoidal Modulated sinusoidal waveform.

2.4 Time Specifications for Pulse Waveform (PULSE)

The general formats for describing a pulse voltage or current source are as follows:

Syntax:

V< Name > < N1 nœuds positifs > < N2 nœuds négatifs>

+ [PULSE (V1 V2 td tr tf pw per)]

I< Name > < N1 nœuds positifs > < N2 nœuds négatifs>

+[PULSE (V1 V2 td tr tf pw per)]

Where:

- **V1:** Rest voltage or current
- **V2:** Pulse voltage or current
- **td:** Delay Time
- **tr:** Rise Time (>0!)
- **tf:** Fall Time (>0!)
- **pw:** Pulse Width

- **per:** Period

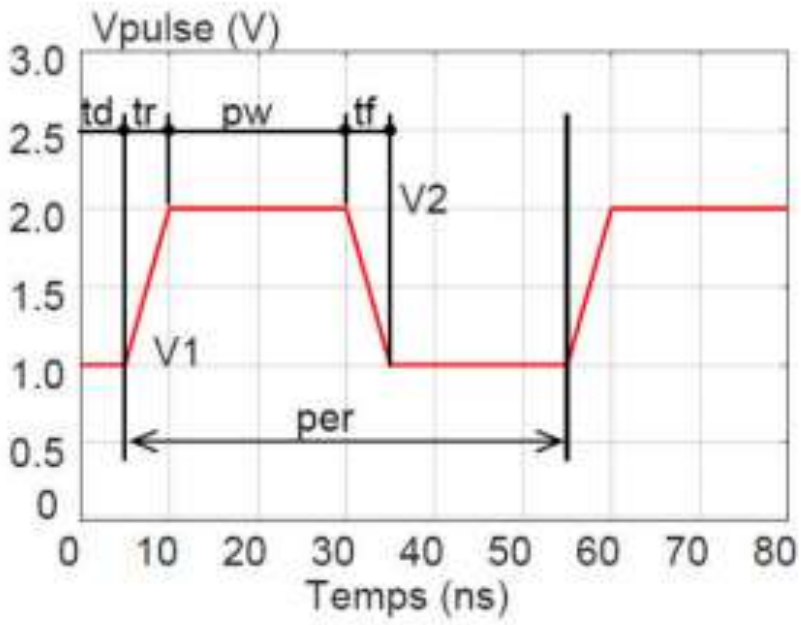


Fig. 4: Pulse waveform

2.5 Time Specifications for Exponential Waveform (EXP)

The general formats for describing an exponential voltage or current source are as follows:

Syntax:

V< Name > < N1 nœuds positifs > < N2 nœuds négatifs>

+[EXP (V1 V2 td1 tc1 td2 tc2)]

I < Name > < N1 nœuds positifs > < N2 nœuds négatifs>

+[EXP (V1 V2 td1 tc1 td2 tc2)]

Where:

- **V1:** Initial value (1st phase)
- **V2:** Asymptotic value (1st phase)
- **td1:** Initial delay
- **τ1:** Time constant (1st phase)
- **td2:** Delay before 2nd phase
- **τ2:** Time constant (2nd phase)

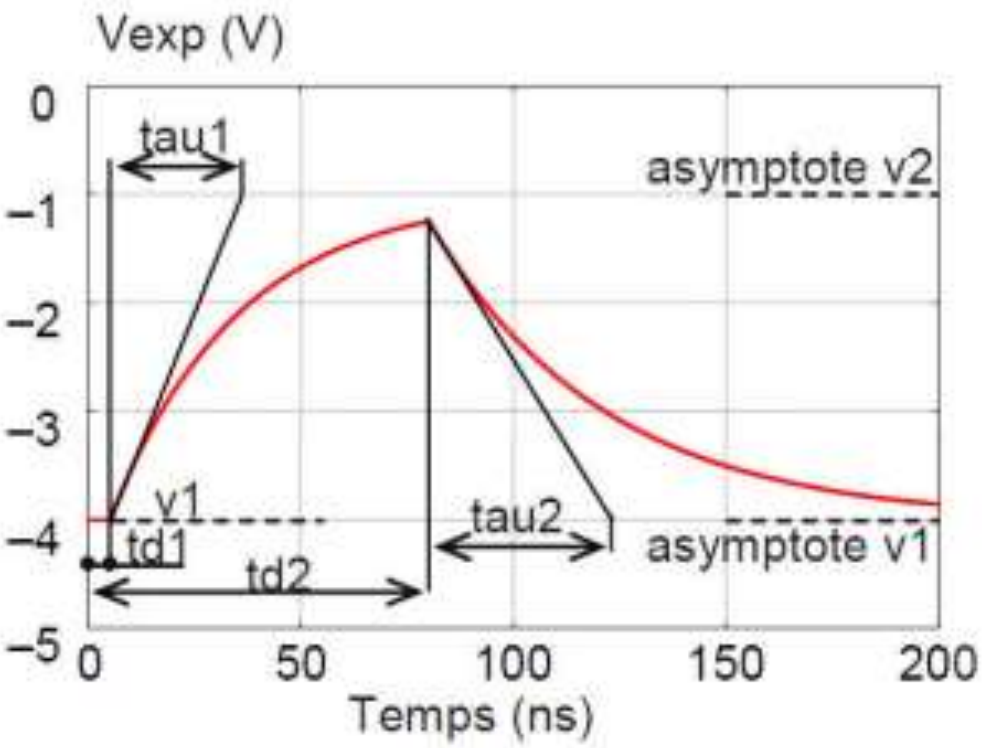


Fig. 5: Exponential waveform

2.6 Time Specifications for Piecewise Linear Waveform (PWL)

The general formats for describing a piecewise linear voltage or current source are as follows:

Syntax:

```
V< Name >    < N1 nœuds positifs > < N2 nœuds négatifs>
+[PWL( t0 v0 t1 v1 t2 v2 . . . tn vn )]
I< Name >    < N1 nœuds positifs > < N2 nœuds négatifs>
+[PWL( t0 v0 t1 v1 t2 v2 . . . tn vn )]
```

Each pair of values (t; v) indicates that at time **t**, the voltage or current of the source is **v** (in volts or amperes). The intermediate values are determined by linear interpolation between the boundary values.

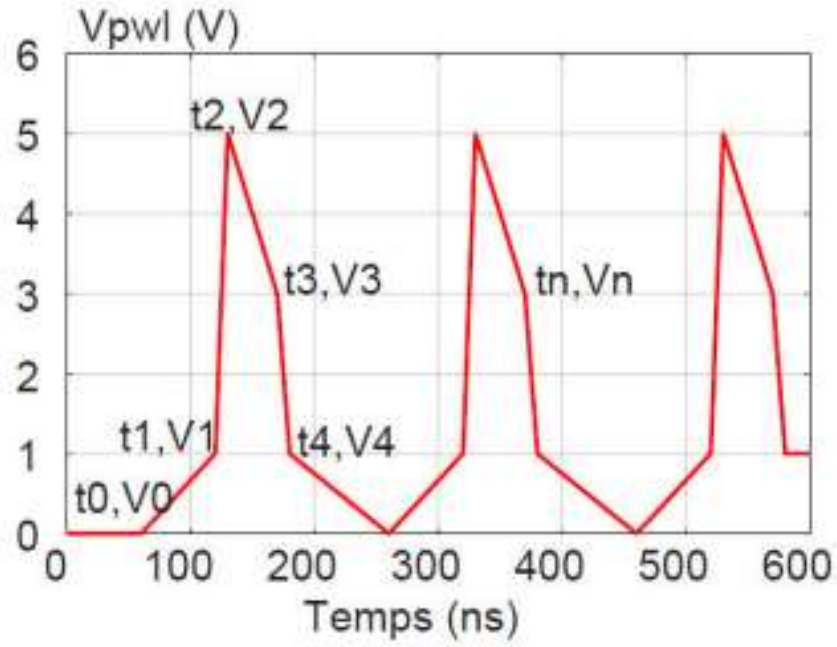


Fig. 6: PWL Form

3. Dependent Sources

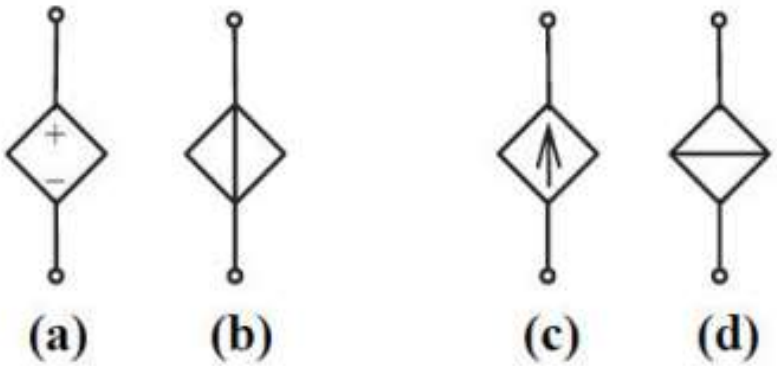


Fig. 7: Symbols of Controlled Voltage Sources (a, b) and Controlled Current Sources (c, d).

3.1 Linear Dependent Sources

As the name suggests, controlled voltage or current sources deliver a quantity that depends on one or more other quantities present in the circuit. A linear controlled source is one whose output is proportional to a single other quantity. The following types are distinguished:

- **E**: Linear Voltage-Controlled Voltage Source (VCVS) → Voltage gain G (unitless)
- **G**: Linear Voltage-Controlled Current Source (VCCS) → Transconductance gm (A/V)
- **H**: Linear Current-Controlled Voltage Source (CCVS) → Trans-resistance R (Ω)
- **F**: Linear Current-Controlled Current Source (CCCS) → Current gain Gi (unitless)

Syntax:

Linear Voltage-Controlled Voltage Source (VCVS) → Voltage gain G (unitless)

```
E< Name >   < N+ > < N- > < NC+ > < NC- > [Val-gain]
```

Linear Current-Controlled Voltage Source (CCVS) → Transconductance gm (A/V)

```
G< Name >   < N+ > < N- > < NC+ > < NC- > [Val-gm]
```

Linear Voltage-Controlled Current Source (VCCS) → Trans-resistance R (Ω)

```
H< Name >   < N+ > < N- > < NC+ > < NC- > [Val-R]
```

Alternate syntax:

```
H< Name >   < N+ > < N- > < VName >  [Val-R]
```

Where:

```
< VName >  <N+ > < N- > 0
```

Linear Current-Controlled Current Source (CCCS) → Current Gain Gi (unitless)

```
F < Name >   < N+ > < N- > < NC+ > < NC- > [Val-Gi ]
```

Alternate syntax:

```
H < Name >   < N+ > < N- > < VNOM >  [Val-R]
```

Where:

```
< VName >  <N+ > < N- > 0
```

with:

- N+ and N- are the nodes of the dependent voltage source.
- NC+ and NC- correspond to the positive and negative nodes of the control element, respectively.
- Val-gain is the Voltage Gain (unitless).
- Val-gm is the Transconductance (A/V).
- Val-R is the Trans-resistance (Ω).
- Val-Gi is the Current Gain (unitless).

When the control element is a voltage source, the parameter **VNOM** corresponds to the name of the voltage source through which the control current flows (as in the case of a current-controlled current source). The positive control current flows from **N+** to **N-** of **VNOM**.

However, if the control element is not a voltage source (such as a resistor, for example), a zero-voltage source must be inserted in the relevant control branch. The control current will then be equal to the current passing through this zero-voltage source and the control element in question.

Example

```
E1 1 0 1 0 1meg
Hlin 6 9 1 0 1.2k
```

- ✓ A voltage source named **E1** connected between terminals 1 and 0, with the control voltage **V = V1 – V0**, where the voltage gain is **G = 1×10⁶**.
- ✓ A voltage source named **Hlin** connected between terminals 6 and 9, with the control current being the one flowing through the voltage source **V = V1 – V0**, where the trans-resistance is **R = 1.2 kΩ**

Noticed

If the control element is not a voltage source (such as a resistor, for example), a zero-voltage source must be added in the relevant control branch. The control current will then have the same magnitude as the current flowing through this voltage source and the control element in question

Example

```
F1  12  5  V2  5
```

F1: A current source controlled by a current, placed between nodes 12 and 5, with the control element being a voltage source **V2**, and the current gain of the source is 5 (dimensionless).

3.2 Non Linear Dependent Sources

The specification of linear dependent sources in PSPICE includes the specification of nonlinear dependent sources. The general format for describing a nonlinear source is given as: Source N+ N- POLY(dim) <Input> <Coefficients>

In this statement:

- The Source parameter represents one of the four dependent sources (E, F, G, and H).
- The dim parameter indicates the degree of the polynomial, and the Coefficients parameter represents the coefficients of the polynomial in ascending order.

- For current-controlled sources, the number of input pairs or names must be equal to the value of the dim parameter.
- Therefore, for **POLY(1),** one input pair or name should be specified, and for POLY(2), two input pairs or two names should be specified.
- The coefficients are listed in ascending order.

Syntax:

E< Name > , G< Name > , H< Name > ou F< Name > < N1 > < N2 >
+POLY(1) < NC+ > < NC- > +< Coefficients>

E< Name > , G< Name > , H< Name > ou F< Name > < N1 > < N2 >
+ POLY(2) < NC+ > < NC- > +< NC'+ > < NC'- >< Coefficients>

NC+ and **NC-** correspond to the positive and negative nodes of the 1st control element, respectively.
NC'+ and **NC'-** correspond to the positive and negative nodes of the 2nd control element, respectively.

POLY(1) allows for the creation of a one-dimensional controlled source (i.e., dependent on only one quantity, either current or voltage **X**) following a polynomial law such as:

$$I' \, or \, V' \; = \; C_0 \; + \; C_1 \times X \; + \; C_2 \times X^2 + \ldots + Cn \times X^{\mathrm{n}}.$$

POLY(2) allows for the creation of a two-dimensional controlled source of order 2 (i.e., dependent on two quantities X and Y from the simulated circuit) following the law:

$$I' \, or \, V' \; = \; C_0 \; + \; C_1 \times X \; + \; C_2 \times Y \; + \; C_3 \times X^2 \; + \; C_4 \times XY \; + \; C_5 \times Y^2.$$

Example

```
1    Gpoly 4 0 POLY(1) V1 V3 2.1 0.3 0 4.0
```

A current source named **Gpoly** is connected between terminals 4 and 0, with a control voltage **V = V1 – V3** and a control law:

$$I' = 2.1\ (A/V) \times V\ (V) + 0.3 \times V + 0 \times V + 4 \times V^2.$$

The coefficients 2.1, 0.3, 0, and 4.0 represent transconductance **gm** (A/V).

Noticed

- ✓ Specifying polynomials of degree higher than 1 is much more complex. Here's an example from a second-degree polynomial of order 3:

$$\boldsymbol{I'}\ or\ \boldsymbol{V'} = \boldsymbol{c_0} + \boldsymbol{c_1} \times \boldsymbol{x} + \boldsymbol{c_2} \times \boldsymbol{y} + \boldsymbol{c_3} \times \boldsymbol{x^2} + \boldsymbol{c_4} \times \boldsymbol{xy} + \boldsymbol{c_5} \times \boldsymbol{y^2} + \boldsymbol{c_6} \times \boldsymbol{x^3} + \boldsymbol{c_7} \times \boldsymbol{x^2y} + \boldsymbol{c_8} \times \boldsymbol{xy^2} + \boldsymbol{c_9} \times \boldsymbol{y^3}.$$

- ✓ In the case of a current-controlled source, if the control element is not a voltage source, a voltage source with a zero voltage value must be placed in the control branch (as in the case of a linear dependent source). The control current will then be the current flowing through this voltage source and the control element in question.
- ✓ It is important to note that the number of control elements (**NC+**, **NC-**, **NC'+**, **NC'-**, ...) must be equal to the value of the **dim** parameter. Any unspecified coefficients will default to zero.

Chapter 4:

Specification of an Op-Amp (Operational Amplifier)

1. Amplifier Model

An ideal operational amplifier (Op-Amp) is simply a linear voltage-controlled voltage source with a very high gain. Therefore, it is quite straightforward to simulate the behavior of this component, as shown in Figure 1. The differential input voltage is applied across a resistance R of 1 GΩ, representing the very high input impedance of the Op-Amp. The output voltage is provided by the linear controlled source E with a gain of 10. The entire setup is built as a sub-circuit for easier future use.

```
AOP ideal
* e+ e- s
.subckt AOP 1 2 3
Ri 1 2 1G
E 3 0 (1,2) 1Meg
.ends AOP
Ve 1 0 DC 1
Xaop 1 0 5 AOP
Ru 5 0 1Meg
.DC Ve -1m 1m 5u
.Probe
.END
```

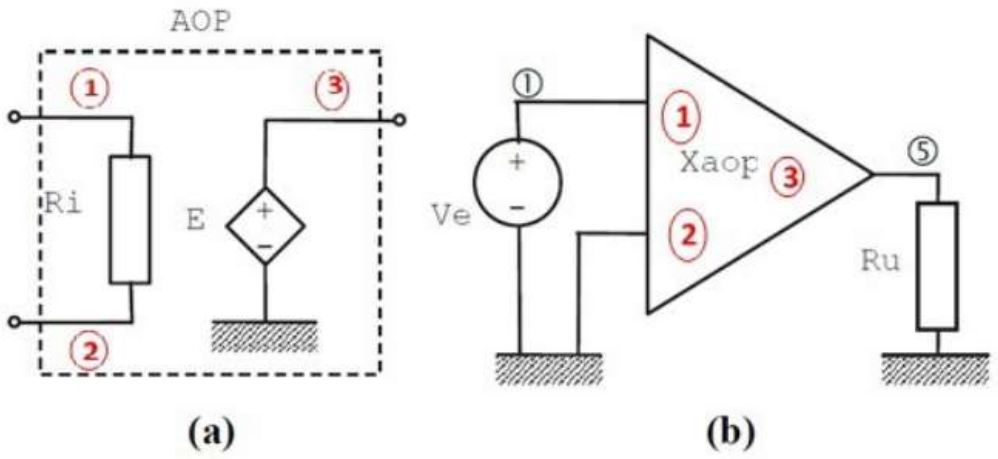


Fig. 1: Simulation of an Ideal Op-Amp

To introduce an operational amplifier (Op-Amp) into a circuit, it is first necessary to define a sub-circuit that represents this amplifier. As shown in Figure 1, where the sub-circuit is used to verify the transfer function of the Op-Amp, the corresponding circuit file is presented below. Note the need to place a load resistance Ru to prevent node 5 from being "floating."

The reader can verify the operation of this circuit using PSpice and observe that an input voltage of 1 V produces an output voltage of 1 MV, which, of course, is entirely unrealistic. To obtain a more realistic behavior that accounts for saturation voltages, the linear controlled source EEE can be replaced with a nonlinear table-based source E, as shown in the circuit file below. The transfer function then becomes that of Figure 2.

```
AOP ideal
* e+ e- s
.subckt AOP 1 2 3
Ri 1 2 1G
E 3 0 TABLE {v(1,2)}=(-20u,-20) (20u,20)
.ends AOP
Ve 1 0 DC 1
Xaop 1 0 5 AOP
Ru 5 0 1Meg
.DC Ve -200u 200u 5u
.Probe
.END
```

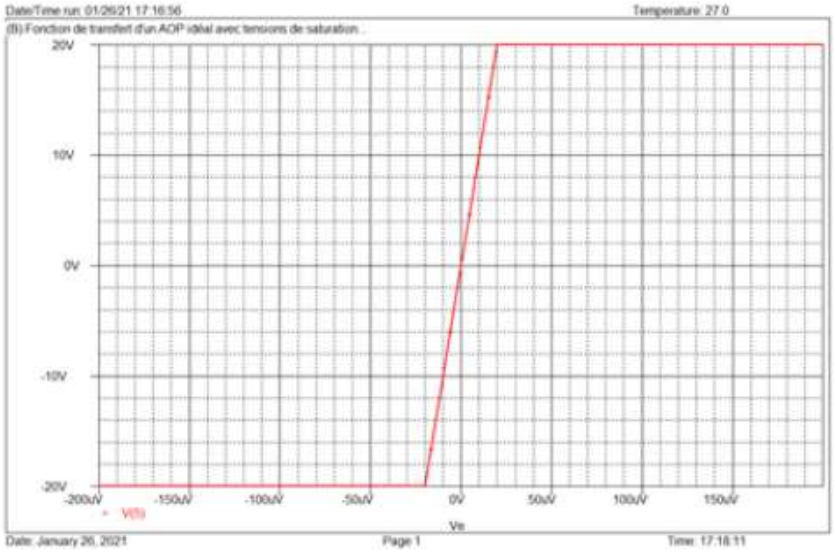


Fig. 2: Transfer Function of an Ideal Op-Amp with Saturation Voltages

2. Application Examples

The Op-Amp component thus created can be used to simulate the behavior of a wide range of circuits, such as an inverting amplifier, integrator, comparator, etc.

Example: Inverting Amplifier

The following configuration (Figure 3) is an inverting amplifier. In this example, note that the non-inverting input of the Op-Amp is grounded, and the expected gain amplification G=−R2/R1 is achieved (Figure 3).

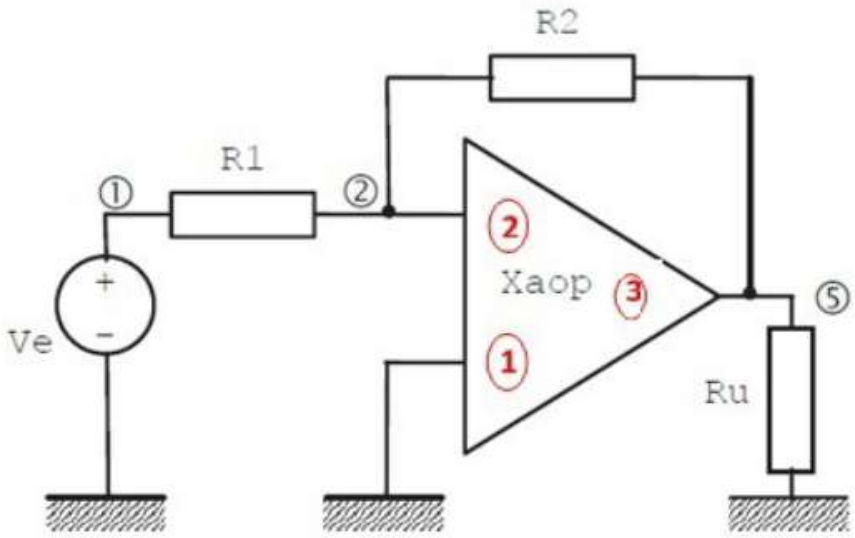


Fig. 3: Inverting Amplifier Circuit with an Ideal Op-Amp

Circuit file for simulating an inverting amplifier with an ideal Op-Amp.

```
Inverseur avec AOP ideal
  * e+ e- s
  .subckt AOP 1 2 3
  Ri 1 2 1G
  E 3 0 TABLE {v(1,2)}=
  + (-20u,-20) (20u,20)
  .ends AOP
  Ve 1 0 DC 1
  R1 1 2 1k
  R2 2 5 10k
  Xaop 0 2 5 AOP
  Ru 5 0 1Meg
  .DC Ve -5 5 50m
  .Probe
  .END
```

Fig. 4: Transfer Function of the Inverting Amplifier

Example: Integrator Circuit Using an Op-Amp

The following figure shows the configuration of an Op-Amp used as an **integrator**.

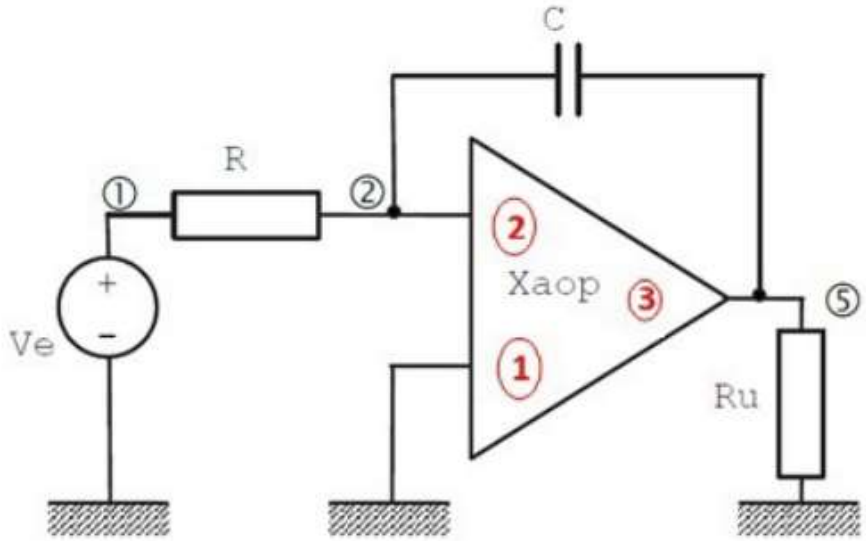


Fig. 4: the configuration of an Op-Amp used as an **integrator**.

The circuit file to simulate the time behavior of this setup is as follows:

```
Integrateur avec AOP ideal
* e+ e- s
.subckt AOP 1 2 3
Ri 1 2 1G
E 3 0 TABLE {v(1,2)}=
+(-20u,-20) (20u,20)
.ends AOP
Ve 1 0
+ pulse(-1 1 0 0 0 0.49m 1m)
R 1 2 1k
C 2 5 100n ic=0
Xaop 0 2 5 AOP
Ru 5 0 1Meg
.tran 10u 5m 0 10u uic
.Probe
.END
```

The input voltage is a square wave with a duty cycle of 1. This means the width of the positive part is equal to half of the period, minus the rise time, which by default is equal to the calculation step size.

At the initial time, the capacitor is discharged (with the initial condition IC = 0), and the simulator takes this into account using the UIC option in the **.TRAN** command.

The result of the simulation is shown in the following figure (Fig. 23), where we observe that the output voltage is indeed proportional to the integral of the input voltage.

This result arises from the fact that the current through the resistor R is the same as the current through the capacitor C. If Vis constant, we obtain:

$$\frac{V_e}{R} = -C\frac{dV_s}{dt} \text{ d'où } V_s = -\frac{1}{RC}\int V_e dt$$

For $V_{\text{in}} = 1\,\text{V}$, $RC = 100\,\mu\text{s}$, and $V(0) = 0$, we should have $V_{\text{out}} = -10t$.

We can verify from Fig. 23 that $V_{\text{out}} = -5\,\text{V}$ at $t = 0.5\,\text{ms}$.

At time $t = 0.5\,\text{ms}$, the output voltage V_{out} is correctly calculated according to the integral of the input square wave voltage.

$$V_s = -\frac{1}{RC}V_e \cdot t + V_s(0)$$

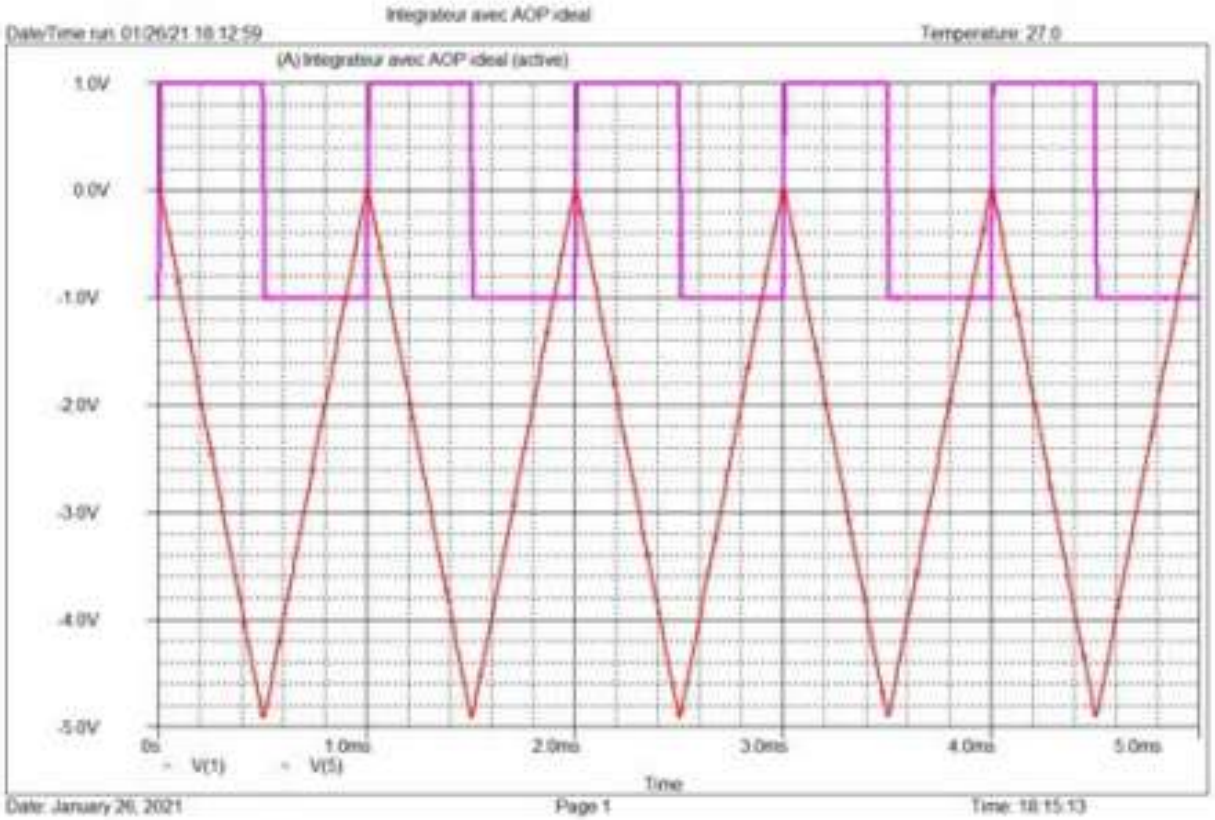


Fig. 5: Time-domain Simulation of the Integrator

Example: Noninverting op-amp circuit

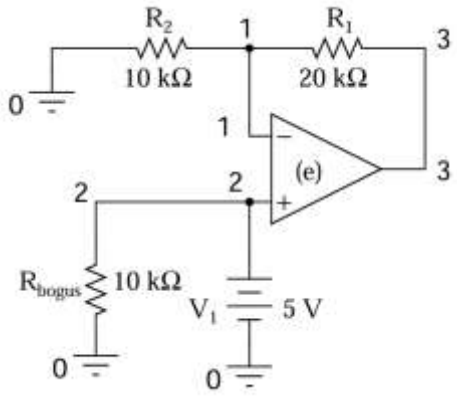


Fig. 6: Noninverting op-amp circuit

Another example of a SPICE quirk: since the dependent voltage source "e" isn't considered a load to voltage source V_1, SPICE interprets $V_1$to be open-circuited and will refuse

to analyze it. The fix is to connect R_{bogus} in parallel with V_1 to act as a DC load. Being directly connected across V_1, the resistance of R_{bogus} is not crucial to the operation of the circuit, so 10 kΩ will work fine. I decided not to sweep the V_1 input voltage at all in this circuit for the sake of keeping the netlist and output listing simple.

Netlist

```
noninverting opamp
v1 2 0 dc 5
rbogus 2 0 10k
e 3 0 2 1 999k
r1 3 1 20k
r2 1 0 10k
.end
```

Output

node	voltage	node	voltage	node	voltage
(1)	5.0000	(2)	5.0000	(3)	15.0000

Example: Instrumentation amplifier

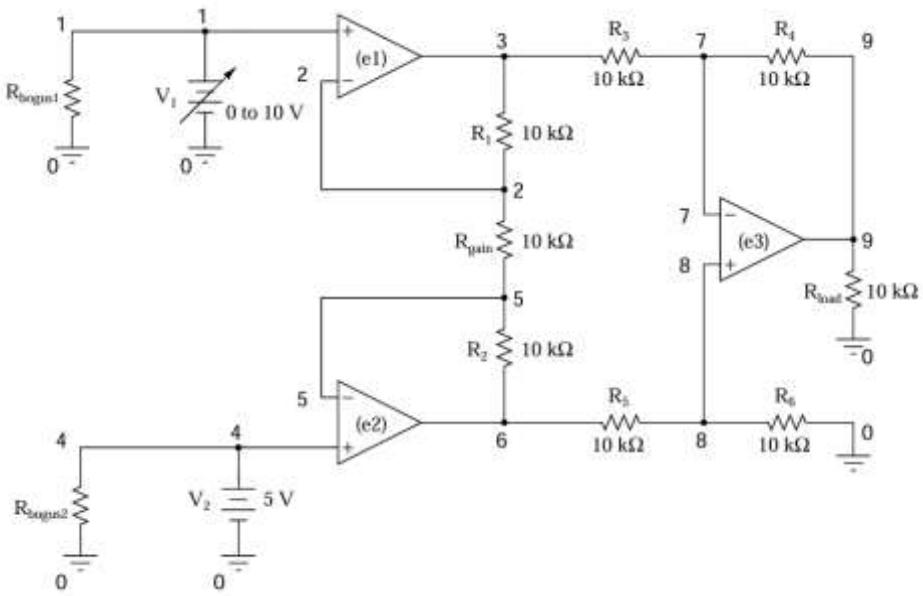


Fig. 7: Instrumentation amplifier

Notetheveryhigh-resistance R_{bogus1} and R_{bogus2} resistors in the netlist (not shown in schematic for brevity) across each input voltage source, to keep SPICE from thinking V_1 and V_2 were open-circuited, just like the other op-amp circuit examples.

Netlist

```
*Instrumentation amplifier
v1 1 0
rbogus1 1 0 9e12
v2 4 0 dc 5
rbogus2 4 0 9e12
e1 3 0 1 2 999k
e2 6 0 4 5 999k
e3 9 0 8 7 999k
rload 9 0 10k
r1 2 3 10k
rgain 2 5 10k
r2 5 6 10k
r3 3 7 10k
r4 7 9 10k
r5 6 8 10k
r6 8 0 10k
.dc v1 0 10 1
.print dc v(9) v(3,6)
.end
```

Output

v1	v(9)	v(3,6)
0.000E+00	1.500E+01	-1.500E+01
1.000E+00	1.200E+01	-1.200E+01
2.000E+00	9.000E+00	-9.000E+00
3.000E+00	6.000E+00	-6.000E+00
4.000E+00	3.000E+00	-3.000E+00
5.000E+00	9.955E-11	-9.956E-11
6.000E+00	-3.000E+00	3.000E+00
7.000E+00	-6.000E+00	6.000E+00
8.000E+00	-9.000E+00	9.000E+00
9.000E+00	-1.200E+01	1.200E+01
1.000E+01	-1.500E+01	1.500E+01

Chapter 5: Analysis and Control Commands

Introduction: A circuit file also contains the commands necessary to execute analyses, set initial conditions, structure the file, or output results. All commands are identified by a keyword that begins with a dot *(.)* and follow a syntax specific to each one. The following two tables summarize the different analysis and control commands that can be performed on a circuit.

Table 1: PSpice Analysis Commands.

DC ANALYSES (NONLINEAR, TIME-INDEPENDENT)	
.DC	Bias Calculation
.OP	Output of bias calculation results
.SENS	Sensitivity Calculation
.TF	DC Transfer Function
DYNAMIC ANALYSES (LINEAR, FREQUENCY DEPENDENT)	
.AC	Small-Signal Analysis
.NOISE	Noise Analysis
TIME DOMAIN ANALYSES (NONLINEAR, TIME DEPENDENT)	
.TRAN	Time Response
.FOUR	Fourier Transform
MULTIPLE ANALYSES	
.STEP	Parametric
.TEMP	Temperature

Tableau 2 : Commandes de contrôle de PSpice.

INITIAL CONDITIONS	
.IC	Sets node voltage for bias calculation
..NODESET	Suggests node voltage for bias calculation
COMPONENT MODELING	
.ENDS	End of subcircuit
..MODEL	Description of a component model
..SUBCKT	Beginning of subcircuit
RESULT OUTPUT	
.PLOT	Text-mode plot
.PRINT	Output in the output file
.PROBE	Creation of the data file for the graphical post-processor
FILE MANAGEMENT	
.END	Marks the end of the circuit file
.INC	File inclusion
.LIB	Reference to a library
.PARAM	Parameter definition
OPTIONS	
.OPTIONS	Sets various limits, control and output parameters

1. Analysis Commands

1.1.The .DC Command (DC Analysis)

The *.DC* command specifies a DC voltage sweep. It has the following general form:

SYNTAX

.DC LINLIN<SRC1> <START1 > < STOP1 > <INC1> [nested sweep specification]

.DC OCTOCT DECDEC <SRC1> <START1 > < STOP1 > <INC1> <number of points>

[nested sweep specification]

.DC OCTOCT DECDEC <SRC1> <START1 > < STOP1 > <INC1> <number of points>

[nested sweep specification]

.DC <variable name> LIST <value>* [nested sweep specification n]

Example

```
1    .DC VIN 0.25 5 0.25
2    .DC VDS 0 10 5 VGS 0 5 1
3    .DC VCE 0 10 0.25 IB 0 10u 1u
4
```

In the first of the previous statements, the source VIN starts with a value of 0.25 V and reaches a value of 5V with an increment step of 0.25V. The specification of the second source *SRC2*, which is optional; and its parameters (*START2*, *STOP2*, *INC2*), produces a sweep for each value of the first specified source, *SCR1*. This is a very practical option when you want to obtain transistor characteristics.

1.2. The .OP Command: Bias Point Calculation

The .OP command calculates the operating point of the circuit. Its general form is very simple:

SYNTAX

.OP

- ✓ It allows the output of detailed information about the bias point. The bias point is calculated even if there is no *.OP* instruction.

- ✓ Without the *.OP* instruction, the only information output concerning the bias point is a list of node voltages. With an *.OP* instruction, the currents and power dissipated by all voltage sources are output
- ✓ Similarly, the bias state and small-signal (linearized) parameters of all controlled sources and all semiconductor components are also output.
- ✓ The *.OP* instruction only commands the output of normal bias point information. It is the *.TRAN* instruction that commands the output of information concerning the bias point of the transient analysis.
- ✓ The bias point is calculated after replacing coils with short circuits and capacitors with open circuits.

1.3. The .TF Command: Transfer Function

The *.TF* command allows calculating the small-signal transfer function of a circuit by linearizing it around a bias point.

It is specified as follows:

SYNTAX

```
.TF <OUTPUT-VAR><INPUT-SRC>
```

- ✓ The parameter *OUTPUT-VAR* is the small-signal output variable.
- ✓ The parameter *INPUT-SRC* is the small-signal input source.
- ✓ The gain between the two variables, as well as the input and output resistances, appear in the output file.
- ✓ The parameter *OUTPUT-VAR* can consist of a potential difference or an output current. In the first case, the syntax is used: *V (N1 <N2>)* which allows specifying the potential difference between nodes *N1* and *N2*.

Example

```
.TF V(5,3) Vin
.TF I(Vload)  Vin
```

1.4. The .AC Command: Small-Signal AC Analysis

The *.AC* instruction is used to calculate the frequency response of a circuit over a frequency range. The arguments LIN, OCT, or DEC are keywords that specify the type of sweep and <number of points> indicates the number of points used during the sweep. General forms:

SYNTAX

```
.AC LIN OCT DEC <number of points> <start frequency> <end frequency>
```

LIN Linear Sweep. The frequency is swept linearly between the start and end frequencies.

<number of points> is the total number of points in the sweep.

OCT Octave Sweep. The frequency is swept logarithmically by octaves. <number of points> indicates the number of points per octave.

DEC Decade Sweep. The frequency is swept logarithmically by decades. <number of points> indicates the number of points per decade.

Example

```
.TRAN 1ns 100ns
.TRAN 1ns 100ns 0ns .1ns
.TRAN 1ns 500ns 100ns
.TRAN 10ns 1us UIC
```

Note

If the *UIC* parameter (use initial parameter) is specified, the program will take into account all the initial conditions of the dynamic elements.

1.5. Relationships between Instructions (.DC .AC TRAN Commands)

Each independent voltage or current source has parameters that specify its use with the different analyses. Figure 24 illustrates the relationships between the description of the source Ve and the analyses performed:

- The *.OP* analysis only considers the *DC=1V* argument to perform the bias point calculation.
- The *.AC* analysis first calculates the bias point from the *DC=1V* argument to establish the linear equivalent circuit and then performs the dynamic analysis from the *AC=1* argument.
- The *.TRAN* analysis only considers the time specification of the source if it exists, here: *SIN(0 100V 50Hz)*. When there is no time specification, it is the *DC=...* argument. that is used .

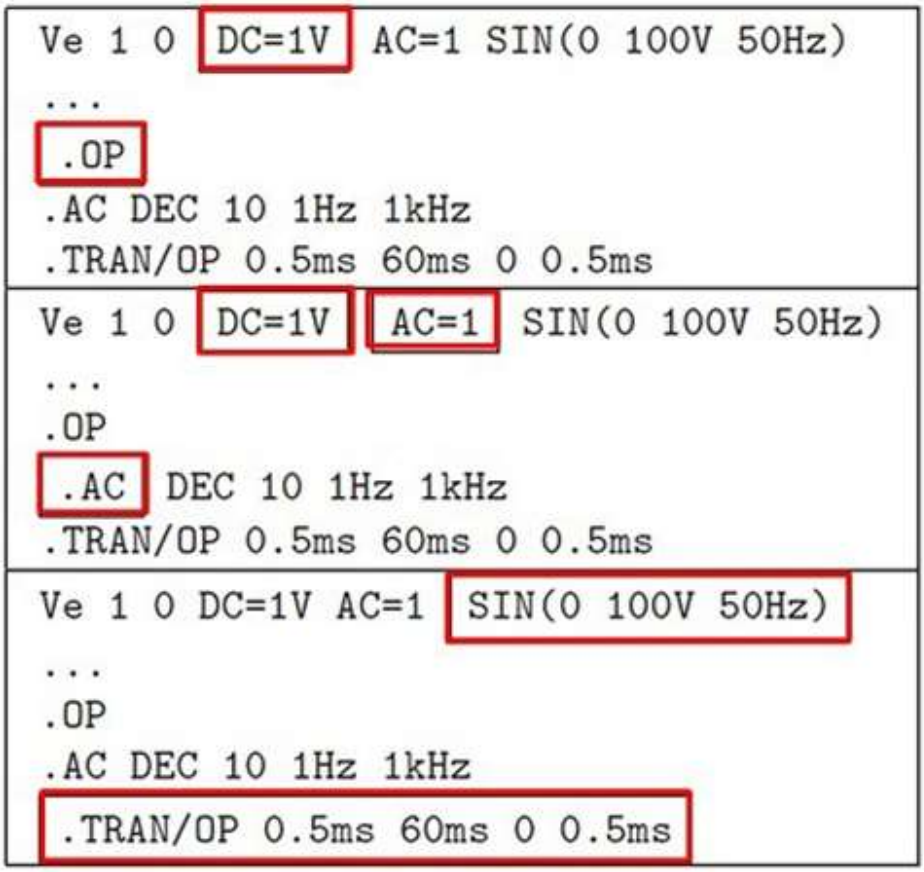


Fig. 1: *Relationships between analyses and source parameters.*

1.6. The .FOUR Command

The .FOUR command performs a Fourier analysis simultaneously with a transient analysis (.TRAN); it provides the results for the first nine terms of the Fourier output. Its general form is:

SYNTAX

```
.FOUR FREQ <output-var1> <<output-var2...>>
```

The parameterFREQ corresponds to the fundamental frequency.

The parameters, output-var1 output-var2 ,..... represent the voltages or currents for which a Fourier analysis is performed.

Example

```
.FOUR 100K V(5)
.FOUR 10K V(5) V(6,9) I(V1)
```

1.7. The .NOISE Command

The .NOISE command is used for circuit noise analysis and works in conjunction with an AC analysis. Consequently, it must absolutely be accompanied by the .AC command in the program to function. It is expressed in the following general form:

SYNTAX

```
.NOISE <OUTPUT-VAR> <INPUT-SOURCE> NUM
```

The parameter OUTPUT-VAR represents the output voltage. The parameter INPUT-SOURCE is the name of an independent voltage or current source that will be the noise source. As for NUM, this parameter represents the value of the printout interval.

At every NUM output frequencies, detailed values of the output are generated. You get announcements such as:

```
.NOISE V(5) Vin 10
.NOISE V(4,5) Isrc 20
```

1.8. The .SENS Command

The .SENS command is used to perform a DC sensitivity analysis. It has the following general form:

SYNTAX

```
.SENS <OUTPUT-VAR1> <OUTPUT-VAR2 ......>
```

- It calculates, one by one, the sensitivity of node voltages or branch currents with respect to variations in the value of each circuit element.
- The parameters OUTPUT-VAR1, OUTPUT-VAR2, ... represent the node voltages or branch currents for which you want to analyze the sensitivity. Their syntax is identical to that of the parameters for the .TF command.

Example

```
1    .SENS V(7) V(4,3) V(11) I(Vcc)
```

1.9. The .TEMP Command

The declaration of the .TEMP command, which is used to set the temperatures during simulation, takes the following general form:

SYNTAX

```
.TEMP T1 <T2 <T3....>>
```

The temperatures are expressed in degrees Celsius. When more than one temperature appears in the command, the program performs the specified simulations for each of these temperatures.

Example

You would enter, for example:

```
1    .TEMP -55 25 126
```

2. Control Commands

2.1. The .OPTIONS Command

The *.OPTIONS* command allows specifying different simulation options and limits. It is declared in the following general form:

SYNTAX

```
.OPTIONS OPT1 OPT2 OPT2 ……. OPT7 = <OPT1-VAL1> <OPT2-VAL2>…….
<OPT7-VAL7>
```

You may encounter a statement like the following:

```
1    .OPTIONS ACCT LIST NODE RELTOL=0.01
```

The order in which the options are specified does not matter. Two types of options are recognized: those that require a value and those that do not. By default, the program considers as inactive *(OFF)* all options that do not require a value. Simply specifying them makes them active *(ON)*.

2.2. The .IC Command

The .IC command sets the initial conditions for the transient analysis. Its specification takes

the general form:

```
.IC V(Node <, Node>) = <VAL>
```

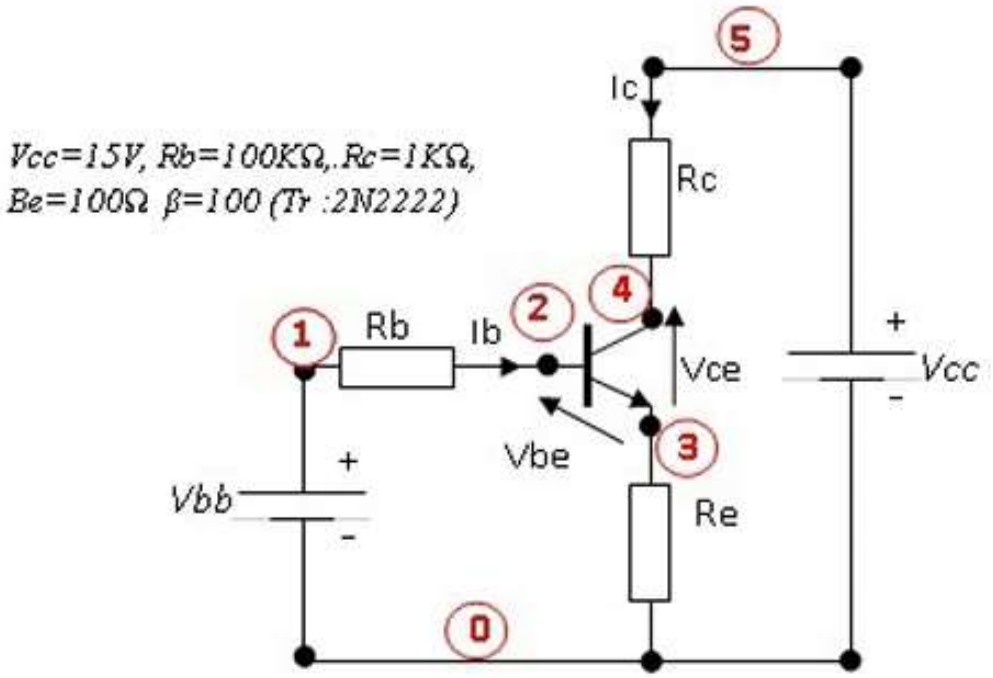
- The parameter **VAL** represents the voltage that will be maintained throughout the bias point calculation.
- Once the bias point is known and the transient analysis has started, the nodes can be "released", meaning the nodes can take on any value.

```
1    .IC V(3,4)=1 V(10)=5 V(14)=-5
```

Note

- The .IC command specifies an initial condition for the **bias point only** and has **no effect on the DC sweep**.

3. Static Simulation (Bipolar Transistor)




```
2    .LIB "NOM.LIB"
3    VBB 1 0 DC 1
4    VCC 5 0 DC 10
5    Q1 4 2 3 Q2N2222
6    RB 2 1 100k
7    RC 5 4 1k
8    RE 3 0 100
9    .DC  VBB 0 5 0.1
10   .print DC V(2,3) V(4,3) IB(Q1) IC(Q1)
11   .OP
12   .probe
13  .end
14 .end
```

The DC analysis command *.DC* and the two output commands *.probe* and *.print* are used:

- To plot the characteristic Ic=f(Vce) (for Vbb=1V and Vcc varying from 0 to 15V with a step of 0.1V)
- To plot the static load line Ic=f(Vce) (with Rc=1kΩ)

The .OP analysis command is used to calculate the operating point Q.

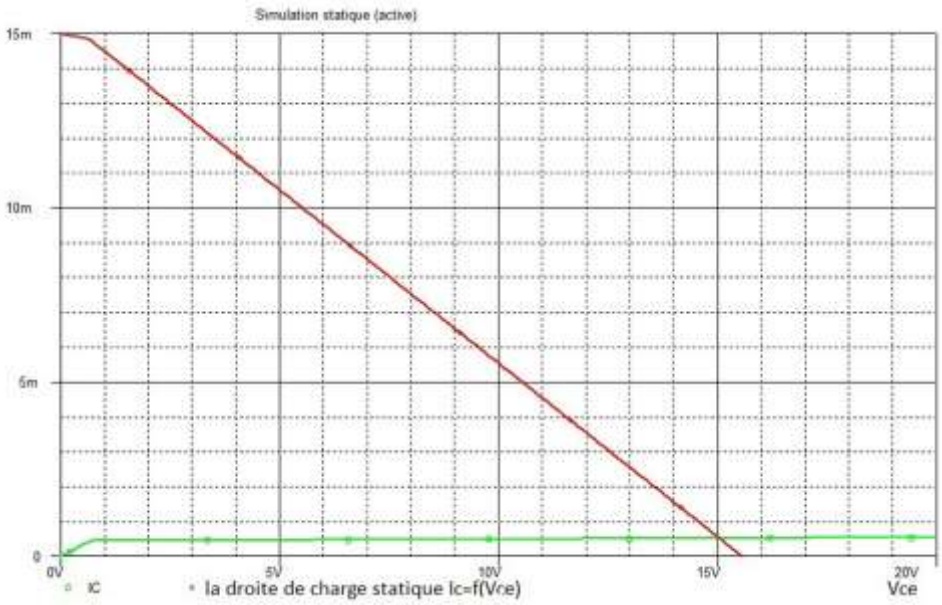


Fig. 2: Ic=f(Vce)

a. Transient Simulation of an Inverting Amplifier

The circuit file for the transient simulation of the inverting amplifier is as follows:

```
* Ampli inverseur
.subckt AOP 1 2 3
Ri 1 2 1G
E 3 0 TABLE {v(1,2)}=(-20u,-20) (20u,20)
.ends AOP
.param amp=1
Ve 1 0 sin(0 {amp} 1k)
R1 1 2 1k
R2 2 5 10k
Xaop 0 2 5 AOP
Ru 5 0 1Meg
.tran 10u 5m 0 10u
.step param amp list 1 3
.Probe
.END
```

The phase inversion between the input and output voltages can be demonstrated through a transient simulation as shown in the circuit file. Note the method of parameterizing the input signal amplitude to perform several identical simulations using the *.PARAM* and *.STEP* commands (consult the documentation for more details on the syntax of these commands).

The result shown in the following figure is then obtained, showing the phase opposition between the input and output voltages as well as the saturation of the output voltage when the input signal amplitude becomes too large.

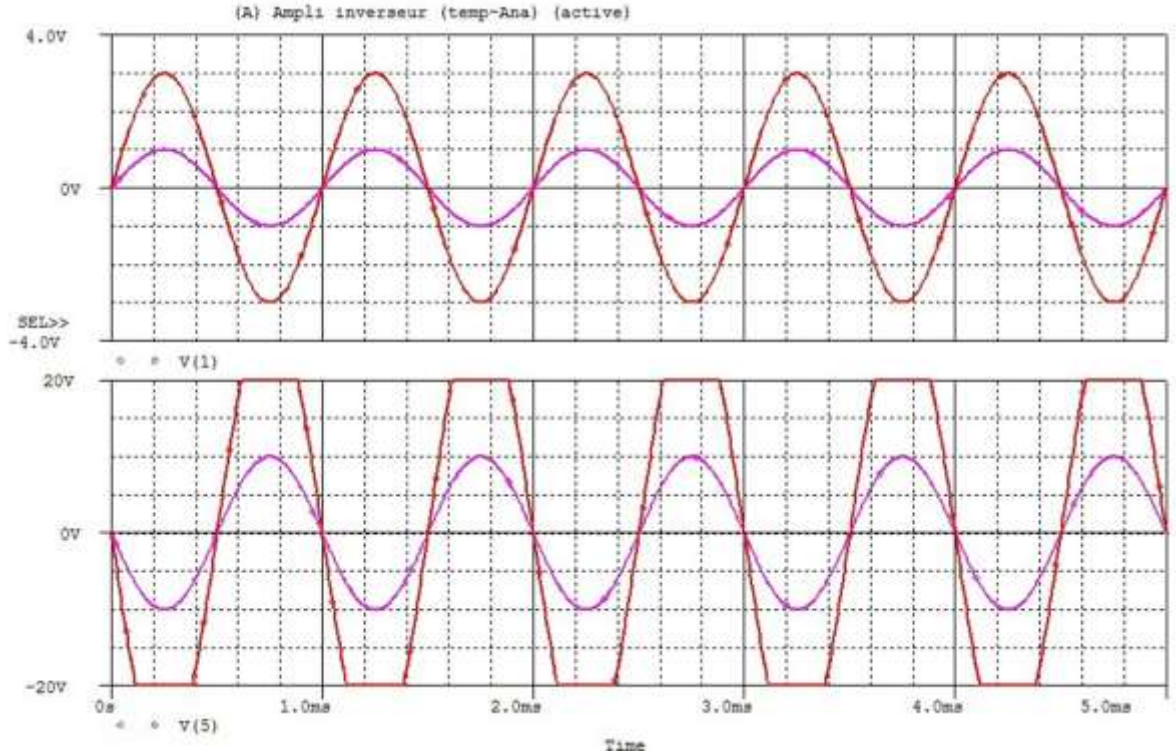


Fig 3: Transient Simulation of the Inverting Amplifier.

Chapter 6: Circuit Simulation Using a Mask Layout Editor

Introduction: This chapter details the essential verification and extraction steps performed after completing a physical layout (mask design) in an Electronic Design Automation (EDA) environment like Cadence Virtuoso and IC Physical Layout Verification Tools. The primary goal is to ensure that the physical layout accurately reflects the intended electrical schematic and is manufacturable. Students will learn how to use a mask layout editor not just for drawing, but as a platform for preparing circuits for fabrication and performance verification.

1. Design Rule Checking (DRC)

1.1. Definition and Purpose

Design Rule Checking (DRC) is an automated verification step in the physical design flow of integrated circuits. Its primary purpose is to ensure that the geometric layout of a chip (the mask design) complies with a set of manufacturing constraints—called Design Rules —specified by the semiconductor foundry.

DRC in SPICE-related contexts (like PCB/IC design) automates verifying a circuit layout against manufacturing constraints (e.g., minimum trace width, spacing) to prevent physical failures, using tools like KLayout or OrCAD to run scripts based on foundry rules, flagging errors like unconnected nets or overlapping shapes for correction before fabrication. It ensures manufacturability and functionality by catching geometric and logical violations in a systematic way, moving from real-time checks during design to batch verification before final output

1.2. Types of Design Rule Checking

Each process technology will have its own set of rules. The number of DRC rules and complexity of rules increases as the manufacturing technology shrinks at advanced nodes. Here are some basic and common types of DRC rules:

- Minimum width
- Minimum spacing
- Minimum area
- Wide metal jog
- Misaligned via wire
- Special notch spacing
- End of line spacing

Table 1: Rule Type and description

Rule Type	Description
Minimum Width	Smallest allowable width for a wire or metal layer.
Minimum Spacing	Minimum distance between two wires or features.
Minimum Area	Smallest permissible area for a given layout feature.
Wide Metal Jog	Rules governing abrupt width changes in metal routing.
Misaligned Via Wire	Ensures vias align correctly with metal layers for connectivity.
Special Notch Spacing	Constraints for spacing around notches to avoid manufacturing defects.
End of Line Spacing	Required spacing at the end of lines to prevent shorts or defects.

1.3. Workflow

1. The layout designer runs the DRC tool (e.g., Assura, Calibre).
2. The tool compares the layout polygons against the rule deck.
3. Violations are highlighted in the layout editor.
4. The designer fixes the violations (resizing, moving, or rerouting).
5. The process repeats until zero violations remain.

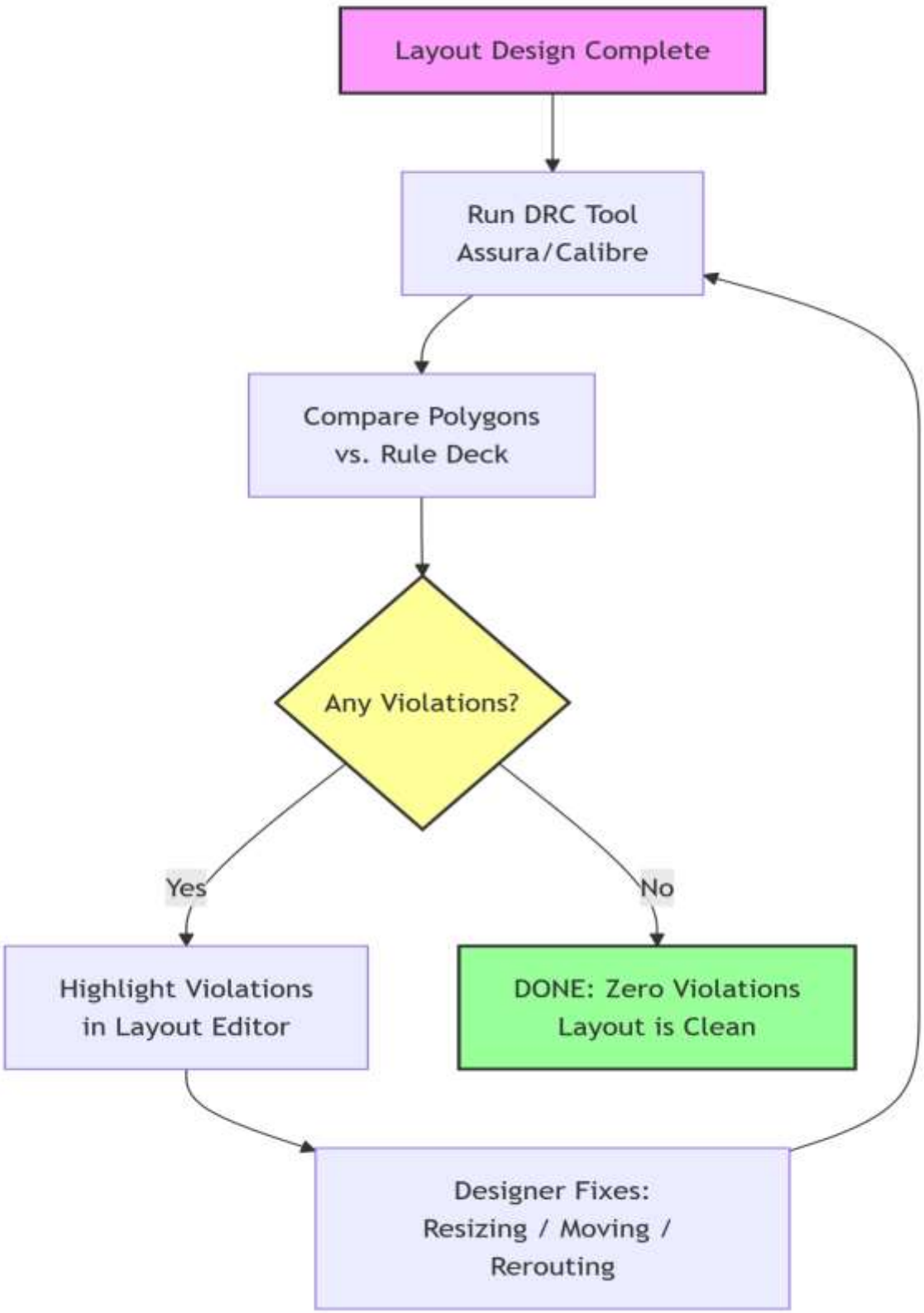


Fig. 1: Block Diagram (Mermaid Code (for GitHub/Markdown))

2. Layout Versus Schematic (LVS)

2.1. Definition

LVS is a verification step that compares the connectivity of the physical layout against the connectivity of the electrical schematic (netlist) .

2.2. Objective:

To confirm that the layout is physically identical to the intended circuit design. A "clean" LVS means the layout contains the exact same electrical connections as the schematic, with no missing or extra components.

2.3. Comparison Metrics:

- Device Count: Does the layout have the same number of transistors, resistors, and capacitors?
- Connectivity: Are the nodes connected exactly as drawn in the schematic?
- Parameters: Are the sizes (e.g., transistor width/length) identical?

2.4. Workflow

1. Extraction: The LVS tool extracts a "layout netlist" from the GDSII data.

2. Comparison: The extracted netlist is compared to the "source netlist" derived from the schematic .

3. Debugging: If mismatches occur (e.g., short circuits, open circuits, parameter mismatches), the layout is corrected.

4. Pass: A successful LVS verification ensures physical implementation matches the design intent .

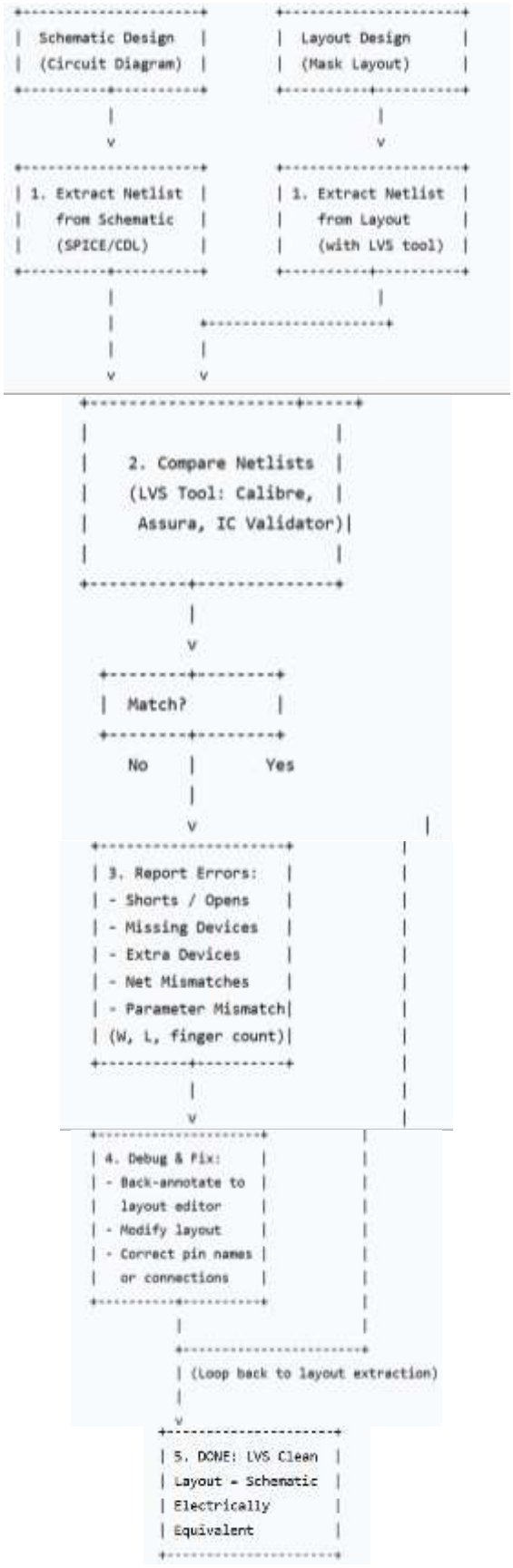


Fig. 2: Simple Flowchart (Text/ASCII)

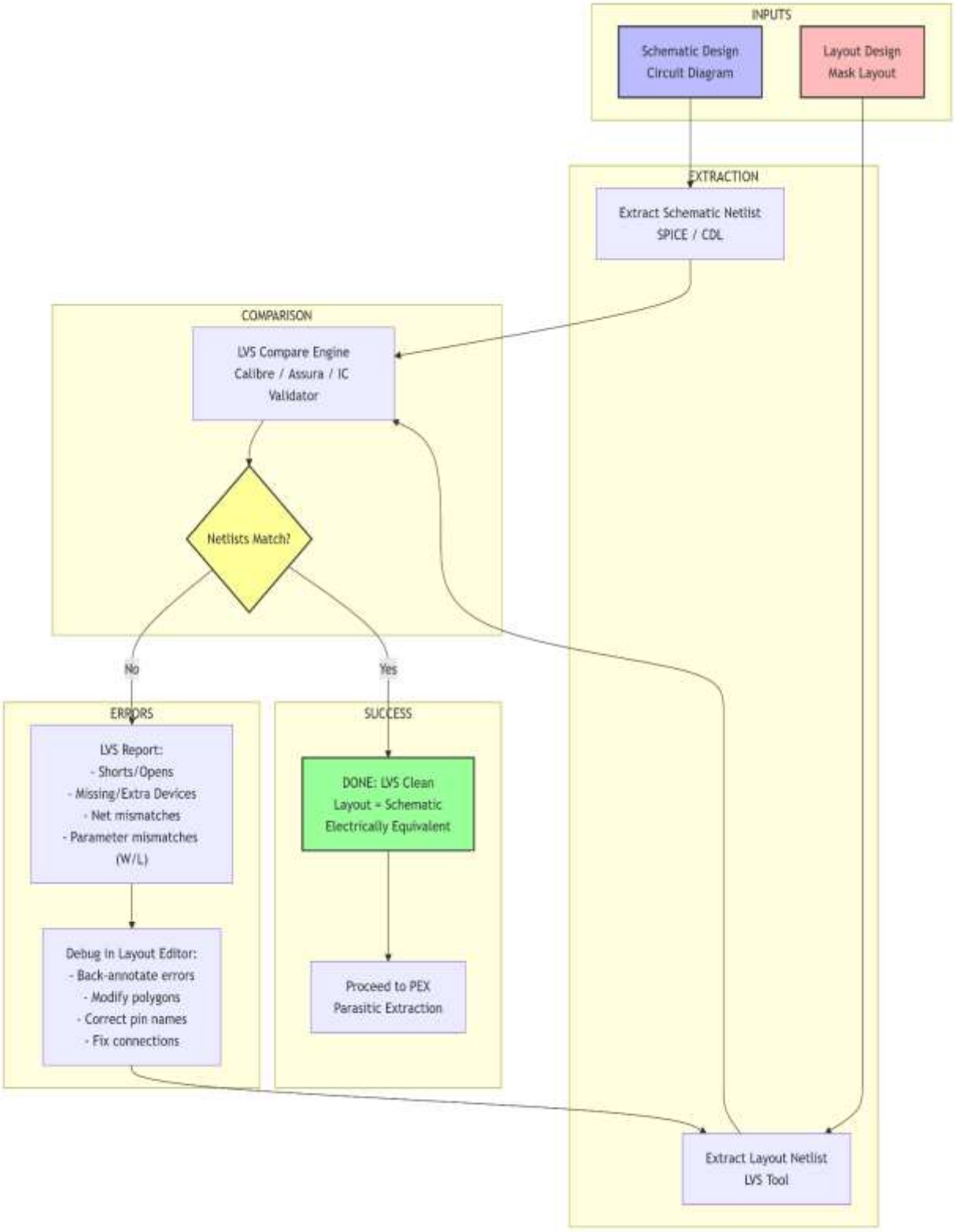


Fig. 3: Block Diagram (Mermaid Code (for GitHub/Markdown))

Table 2: Key Differences: DRC vs. LVS

Aspect	DRC	LVS
Compares	Layout vs. Design Rules	Layout vs. Schematic
Inputs	Layout polygons + Rule deck	Schematic netlist + Layout netlist
Errors	Width, spacing, density violations	Shorts, opens, missing/extra devices, net mismatches
Fix requires	Moving/resizing polygons	Correcting connectivity or device parameters
Output	DRC clean	LVS clean (electrically equivalent)

3. Parasitic Components Extraction of (PEX)

3.1. Definition

After DRC and LVS are clean, the tool extracts parasitic resistances (R) and capacitances (C) inherent to the physical layout .

3.2. Objective

To account for the "hidden" components that affect circuit performance (speed, power, signal integrity) which are not present in the ideal schematic.

3.3. Types of Parasitics

- Coupling Capacitance: Between adjacent metal wires.
- Substrate Capacitance: Capacitance from wires to the silicon substrate.
- Via/Contact Resistance: Resistance introduced by interlayer connections.
- Wire Resistance: Resistance of the metal traces.

3.4. Output Formats:

- DSPF / SPEF: Standard formats for timing analysis.
- Extracted View: A schematic view containing the original devices plus the extracted parasitic R and C .

3.5. Simulation Impact (Post-Layout):

- The ideal schematic simulation (Pre-Layout) is compared to the extracted simulation (Post-Layout).
- Result: Parasitics typically increase delay and reduce bandwidth.
- Example: In a CMOS inverter, delay increased from 115.1 ps (ideal) to 122.4 ps (post-layout) due to extracted parasitics .

Table 3: Key PEX Parameters and Typical Values

Parasitic	Symbol	Typical Range	Impact
Wire Resistance	R	0.01 - 100 Ω/sq	IR drop, delay
Wire Capacitance	C	0.05 - 0.2 fF/µm	Delay, power
Coupling Capacitance	Cc	0.01 - 0.1 fF/µm	Crosstalk, noise
Inductance	L	0.1 - 1 nH/mm	Ringing, EMI (RF/analog)

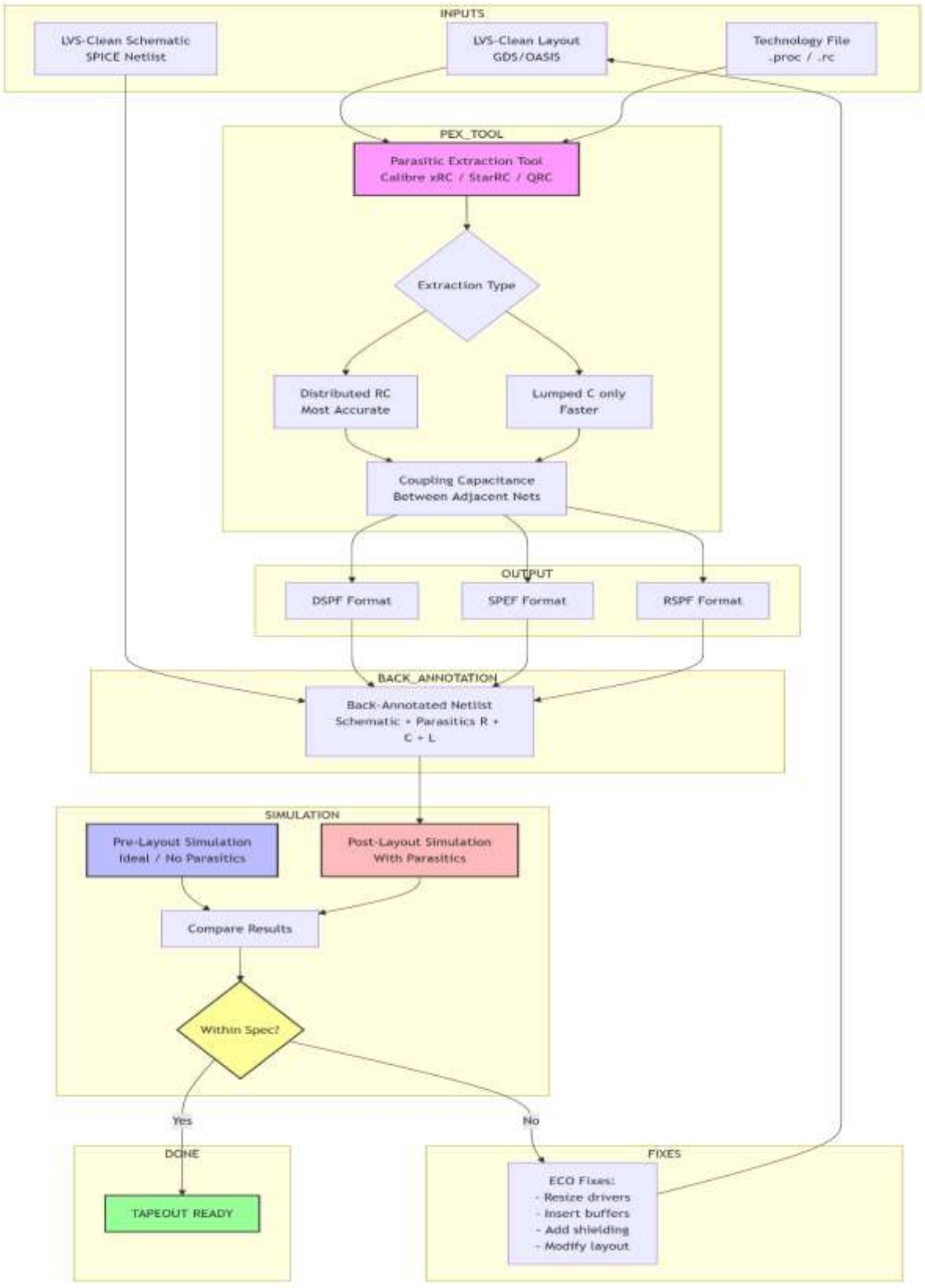


Fig. 4: Block Diagram (Mermaid Code (for GitHub/Markdown))

4. GDSII File Generation (Tapeout)

4.1. Definition

Definition: The final step where the layout database is exported into a standard file format (GDSII) to be sent to the foundry for mask fabrication .

4.2. Objective

To produce the final "tapeout" data that defines the photomasks used in the lithography process.

4.3. File Details:

- Format: Binary or ASCII (Stream Format).
- Layers: Geometric shapes assigned to specific layer numbers and datatypes (e.g., Layer 31: Metal 1, Datatype 0: Drawing).
- Hierarchy: Retains the cell structure of the design (top-level cell calling sub-cells).

4.3. Generation Settings:

- Map File: A configuration file defining which Virtuoso layers map to which GDSII layer numbers .
- Stream Out: The command or menu option used to export the data.
- Log File: Generated to verify that no data was lost or corrupted during export.

Final Output: The generated `.gds` or `.gdsii` file is the final deliverable. It is archived and sent to a mask shop where it is converted into hardware .

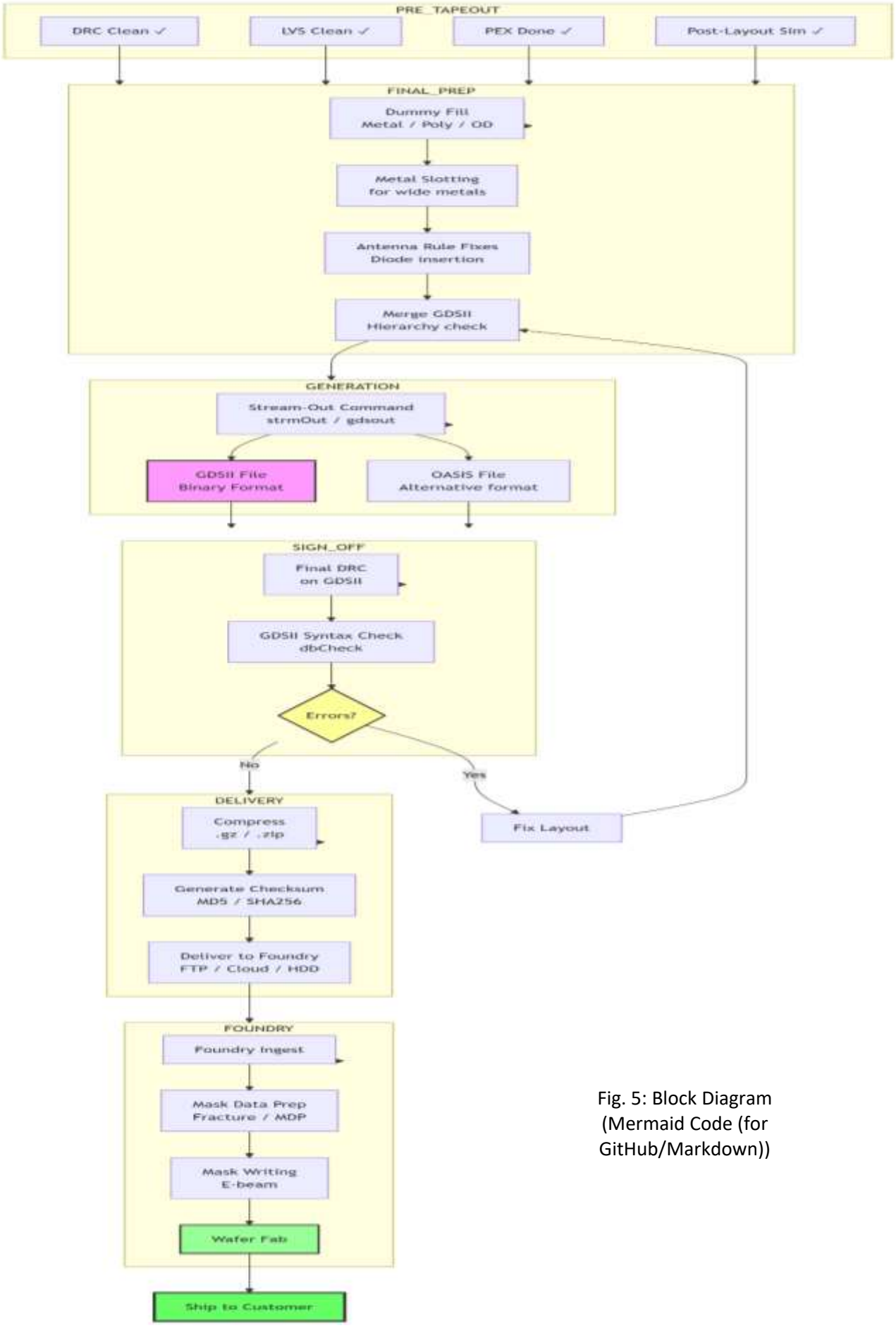


Fig. 5: Block Diagram (Mermaid Code (for GitHub/Markdown))

5. Summary of the Workflow

The following table summarizes the four critical steps executed in the layout editor:

Table 4: critical steps executed in the layout editor

Step	Abbreviation	Function	Success Criteria
Design Rule Check	DRC	Verifies manufacturing constraints (width, spacing) .	Zero rule violations.
Layout vs. Schematic	LVS	Verifies electrical connectivity matches the schematic .	Netlists match perfectly.
Parasitic Extraction	PEX	Extracts hidden R & C from the geometry .	Generation of a netlist with parasitics.
GDSII Generation	GDS Out	Exports final layout to industry standard format .	Creation of a binary .gds file.

Series of Directed Work No. 1

(Specification of Passive Elements)

Exercise 1:

1) Describe the following circuits in SPICE program:

   a- 5th order Butterworth low-pass filter based on the following passive components:

      - A resistor named R_{charge} with a value of 1KΩ

      - Three inductors named L1, L2, and L3 with values of 30.7mH, 27.5mH, and 6.15mH, respectively

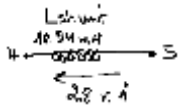
      -Two capacitors named C1 and C2 with values of 0.033 µF and 0.018 µF, respectively.

   b- Using the same previous components to describe two filters: a band-pass filter and a second-order high-pass filter.

   c- A transformer with two windings: To simulate a transformer in SPICE, you must specify the primary and secondary windings as inductors. A factor of k = 0.99 should be specified (to relate LS with LP).

Exercise 2:

Complete the following table:

Specification of components/electrical diagrams	SPICE declaration
A resistor named Rtemp with a value depending on the temperature Rtemp = 1000 [1 + 0.003 (T − 27) + 0.01 (T − 27)2] placed between nodes 7 and 0	
A nonlinear inductor named LNL placed between nodes 5 and 0, with the coefficients L0, L1, and L2 of the polynomial describing the inductance value being: 1, 2.5, and 2, respectively.	
	`Cpar electr1 electr2 +10u IC= 3`
	
An inductor named L3 of 0.1H placed between nodes 11 and 0 with an initial current of 20mA.	
	`Llink 3 1 0.10H IC= +19.8M`

Series of Directed Work No. 2

(Specification of Active Elements, Analysis Controls)

Exercise 1:

Draw the electrical diagrams for each SPICE program.

Circuit 1	Circuit 2	Circuit 3
R1 1 2 1K R2 3 0 8.2MEG R3 4 0 330 C1 4 0 100U C2 2 5 10U C3 3 6 .003U J1 2 3 4 +J2N3819 ROUT 5 0 100K	Rin IN gate 1k Rg grille 0 2K Rout cathode out 100k Dprot drain cathode + DN4148 Mmos drain gate 0 +0 MIRF630	COSC 9 4 0.001 U RS2 10 9 50 RE1 4 5 100 RE2 5 0 1K C2 5 0 0.001U CT 2 0 1224P LT 1 2 100U LS 11 0 1.75U KTS LT LS 0.9 Rl 11 0 1K Q1 2 3 4 QMOD

Exercise 2: Consider the circuits in the figure below:

- Number the nodes of the three circuits
- translate the three diagrams into PSPICE language

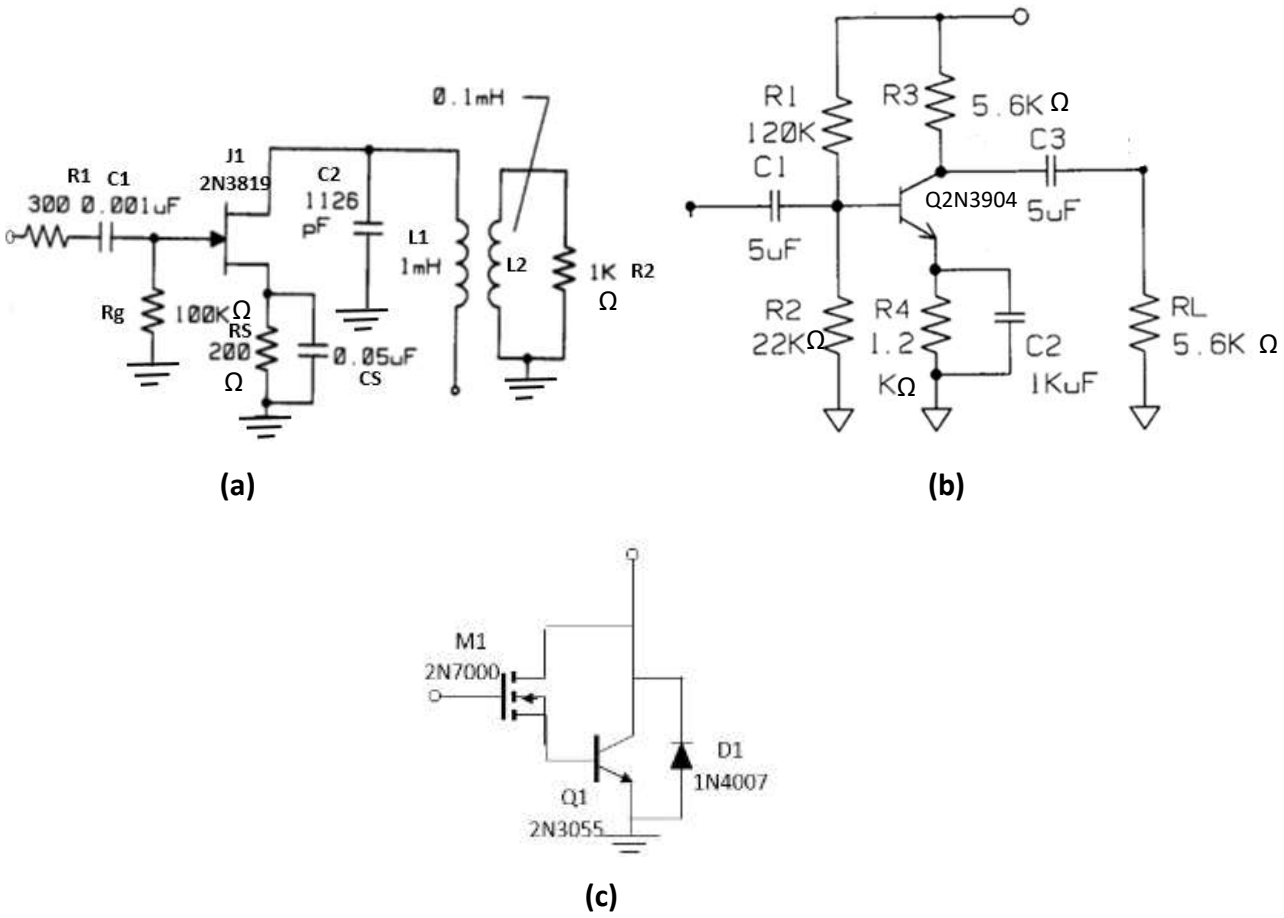


Figure 1: Circuits with passive and active components

Exercise 03: Explain the role of the following instructions:

- `.PROBE`
- `.OP`
- `.DC VIN LIST 0 5 -10 15 20 -25 -5 10 -15 -20 25`
- `.DC VDS 0 10 5 VGS 0 5 1`
- `.DC VCE 0 10 0.25 IB 0 10u 1u`
- `.AC OCT 21 1 1MEG`
- `.TRAN 1ns 100ns 0ns .1ns`

Series of Directed Work No. 2

(Specification of independent sources)

Exercise 1:

Consider the following independent DC sources:

1. Express these types of sources in SPICE language.
2. If we assume that the DC sources (a) and (c) are:

- sinusoidal sources of current and voltage, with amplitudes of 10 A and 5 V respectively, a frequency of 1 kHz, and a delay of 10 s, provide the resulting SPICE declaration expressions.
- variable frequency sinusoidal sources, with a damping factor θ and a phase shift ϕ equal to ($5*10^7$, π/4), ($5*10^7$, 0) respectively, provide the SPICE declaration expressions.

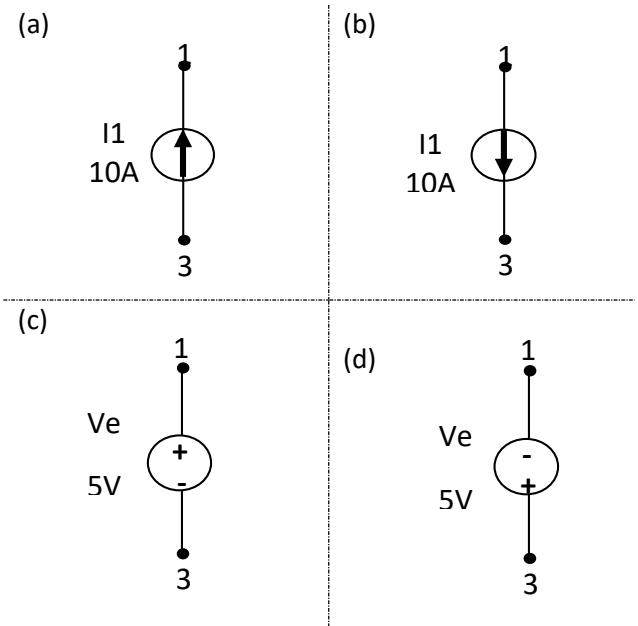
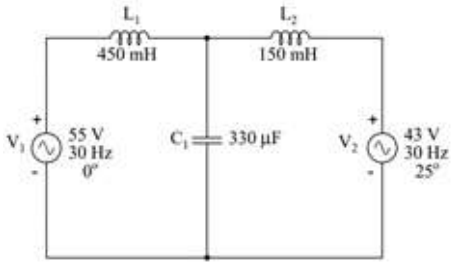
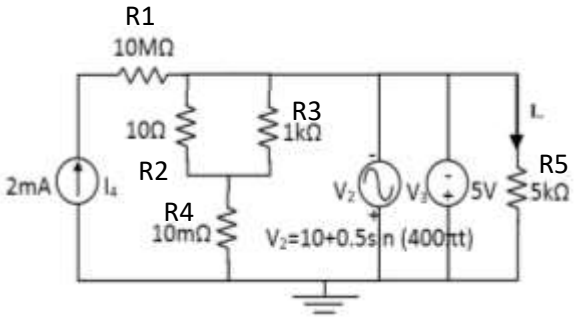


Figure. 1 : examples of independent sources

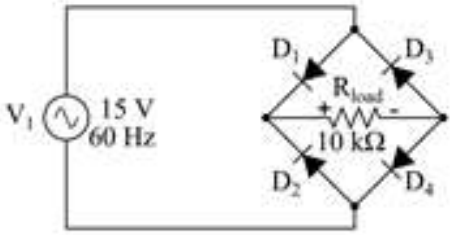
Exercise 2: Consider the circuits in Figure 2.



Circuit (a)



Circuit (b)



Circuit (c)

Figure. 2: Example of different circuits

1. Number the nodes of circuits (a), (b), and (c).
2. What is the description of the current sources and circuits (a) and (b) in SPICE language?
3. What is the description of circuit (c) in SPICE language? Use the **.model** instruction that specifies 'D' as a generic diode model for mod1 in its specification.
4. Write the necessary commands to perform a transient analysis with a time step of 5 ms and a final time of 25 ms

Exercise 3:

Consider the following circuits: We want to perform a static simulation of the circuit in PSPICE.

1. Number the nodes of the circuit.
2. What is the description of the circuit in SPICE language?

Write the necessary commands to perform a DC analysis of circuit (b) for Vbb = 1V while varying Vcc from 0 to 15V with a step of 0.1V.

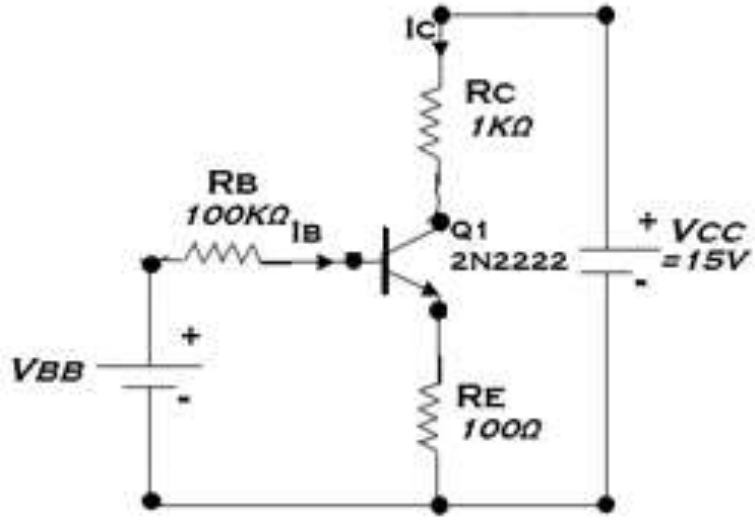


Figure. 3: circuit based on bipolar transistor

Exercise 4:

Consider the waves illustrated in Figure 4:

1. Provide the SPICE declaration expression for these waves using the PULSE and SIN commands.
2. Translate the following SPICE declarations of the waves into graphs:

- Ve 1 0 pulse (1 3 10n 5n 5n 20n 50n)
- Vin 1 0 pulse (-1 5 10n 1f 1f 5n 20n)

3. Redo the declaration of the expressions for Ve and Vin using the PWL (piecewise linear) command over a period T.

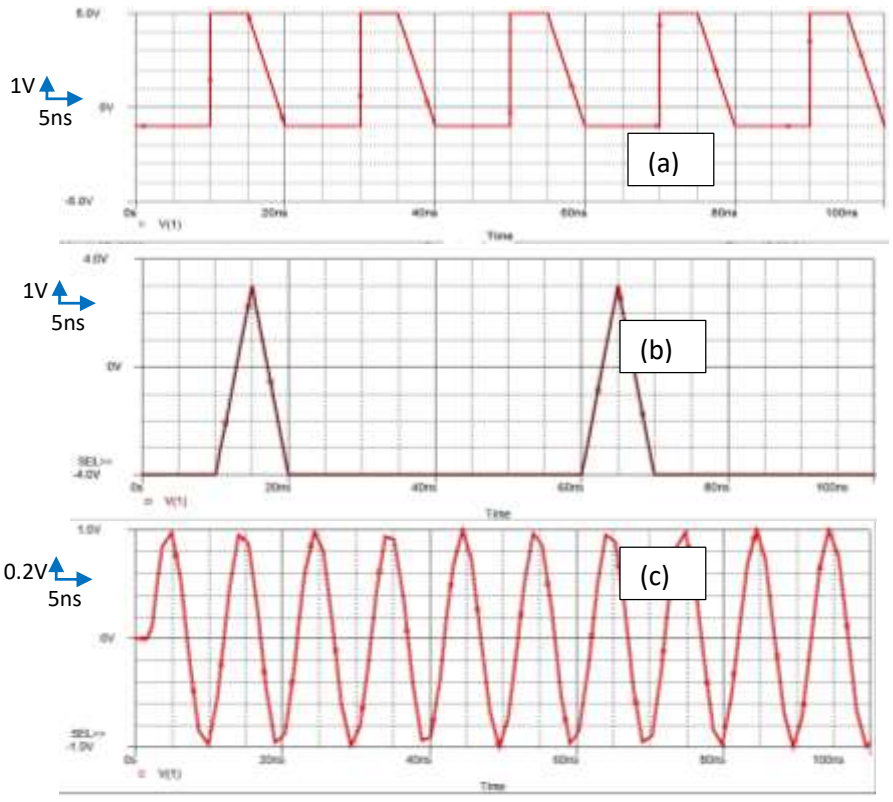
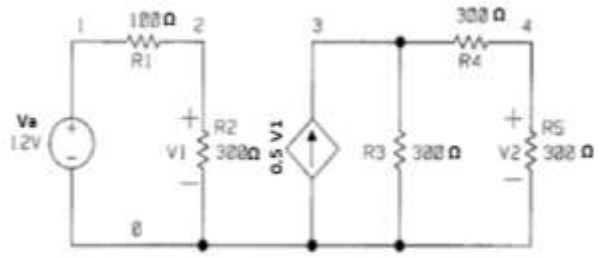


Figure. 4 : Wave Formes

Series of Directed Work No. 3

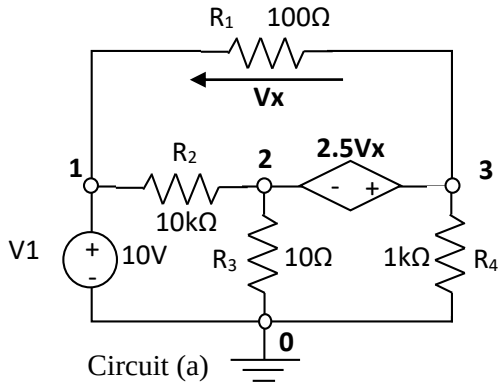
(Specification of Linear and Nonlinear Dependent Sources of independent sources)

Exercise 1: Consider the circuit in the following figure:
We wish to perform a PSPICE simulation of the circuit, where the control element is a resistor.

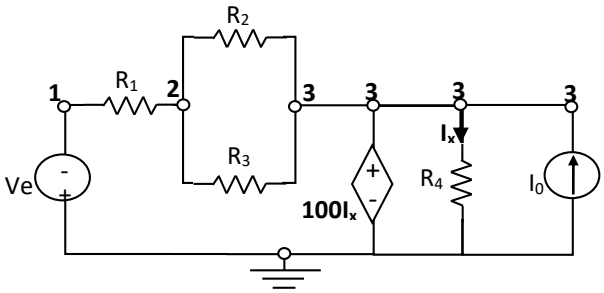


1. What is the type of controlled source and the type of control quantity used in this circuit?
2. What does the coefficient multiplied by the control quantity represent? Provide their units.
3. What is the description of this circuit in PSPICE language?

Exercise 2: Consider the circuits in Figure 2.



Circuit (a)



Circuit (b)

Figure 3: Controlled Linear Voltage Sources

We wish to perform a PSPICE simulation of circuits (a, b), where the control element is a resistor.

1. What is the type of controlled source and the type of control quantity used in each circuit?
2. What does the coefficient multiplied by the control quantity represent? Provide their units.
3. What is the PSPICE description of circuit (a)?

Exercise 3: Draw the electrical diagrams for each SPICE program.

```
Circuit.1 (BJT)
Vbe B 0 0.7
Vce C 0 5
Rh11 B 1 100
Rh22 C 0 2k
Ee 1 0 C 0 1e-3
Ge C 0 B 0 100
```

```
Circuit. 2 (JFET)
Vgs G S DC 5
Vds D S DC 2
Rgs G S 100
Rd D S 10k
Rds D S 10
Gfet D S G S 50m
```

```
Circuit.3 (CMOS)
Vin  1  0  Pulse  0  1
+50u 0 0 100u 150u
Vout 2 0 DC
RNout 2 0 1k
RPout 2 0 2k
Gmn 0 2 1 0 0.8
Gmp 0 2 1 0 0.2
```