WaSabi

User manual

Bonomi Enrico Pezzoli Matteo Scandella Matteo

March 25, 2016

Contents

Introduction 1

1 Basic usage 2

1.1 Launching the tool . . . 2

1.2 Compiling a WaSabi source file . . . 2

1.3 Exporting to a SPICE file . . . 4

2 Language 5

2.1 Language introduction . . . 5

2.2 The program structure . . . 5

2.3 The libraries section . . . 6

2.4 The constants section . . . 6

2.4.1 Value declarations . . . 7

2.4.2 Source declarations . . . 7

2.4.3 Model declarations . . . 7

2.5 The components section . . . 7

2.5.1 Resistance . . . 8

2.5.2 Capacitor . . . 9

2.5.3 Inductor . . . 10

2.5.4 Diode . . . 10

2.5.5 Bipolar Junction Transistor (BJT) . . . 11

2.5.6 Metal-Oxide-Semiconductor FET (MOSFET) . . . 12

2.5.7 Junction FET . . . 13

2.5.8 Generic model . . . 14

2.5.9 Voltage source . . . 14

2.5.10 Current source . . . 15

2.6 The simulation directives section . . . 16

2.7 List of keywords . . . 16

2.8 Semantic errors . . . 18

2.8.1 Constants with the same name . . . 18

2.8.2 Constant not defined . . . 19

2.8.3 Duplicate library . . . 19

2.8.4 Floating node . . . 19

2.8.5 No components in the circuit . . . 19

2.8.6 No GND reference . . . 19

2.8.7 Parallel voltage exception . . . 19

2.8.8 Short circuits . . . 20

2.8.9 Using the wrong constants . . . 20

2.8.10 Declaring components with value at zero . . . 20

3 Examples 21

Introduction

WaSabi (Wow! A Spice Amazingly Barbaric Implementation) is a tool used to generate SPICE[I]-formatted files using a somewhat more verbose and intuitive language. SPICE (Simulation Program with Integrated Circuit Emphasis) is an open source program used for circuit simulation; it was developed in 1975 by the Electronics Research Laboratory of the University of California. SPICE bases its operating principle on a text-based file, the netlist, that is translated into a set of nonlinear differential equations to be solved. The purpose of the WaSabi language is to offer a new approach at the generation of the netlist file.

About and feedback

This language provides an easier access to SPICE core functionalities in order to facilitate the creation of SPICE-compatible files that can be imported and simulated. It was created by three students of the University of Bergamo, Italy, as a project for the «Languages and Compilers» class. To submit issues or give feedback on the project please refer to the Github pageGithub page[II].

[I]http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/
[II]https://github.com/Pezz0/WaSabi

1 Basic usage

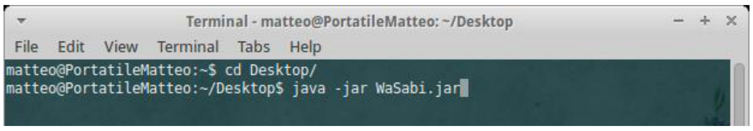
This chapter will be dedicated to the understanding of the basic usage of the tool; it will cover most of the use cases, using examples to guide the user in its first uses of the WaSabi language.

1.1 Launching the tool

In its current state, the tool is distributed using a *.jar* file. In order to run the tool, a Java release[I] must be installed on the machine; if the computer does not have Java installed, please refer to the Java download page.

Given that the computer has Java installed, it can run the tool; first of all the user must navigate to the *.jar* file location and launch using the *java* command with the *jar* argument, such as:

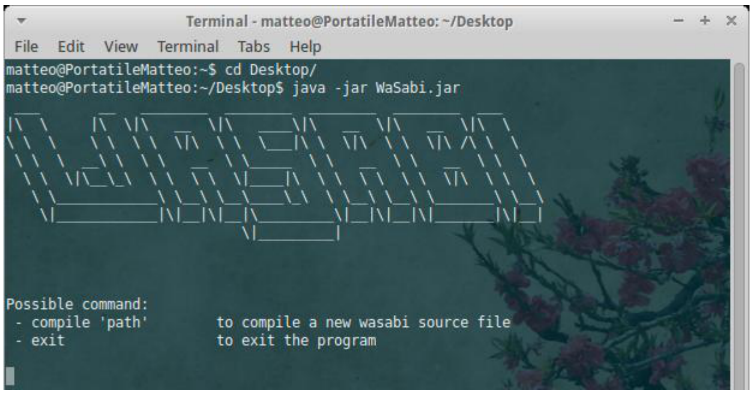
```
$ java -jar WaSabi.jar
```



```
matteo@PortatileMatteo:~$ cd Desktop/
matteo@PortatileMatteo:~/Desktop$ java -jar WaSabi.jar
```

1.2 Compiling a WaSabi source file

After starting the tool the command windows should look like the one below:

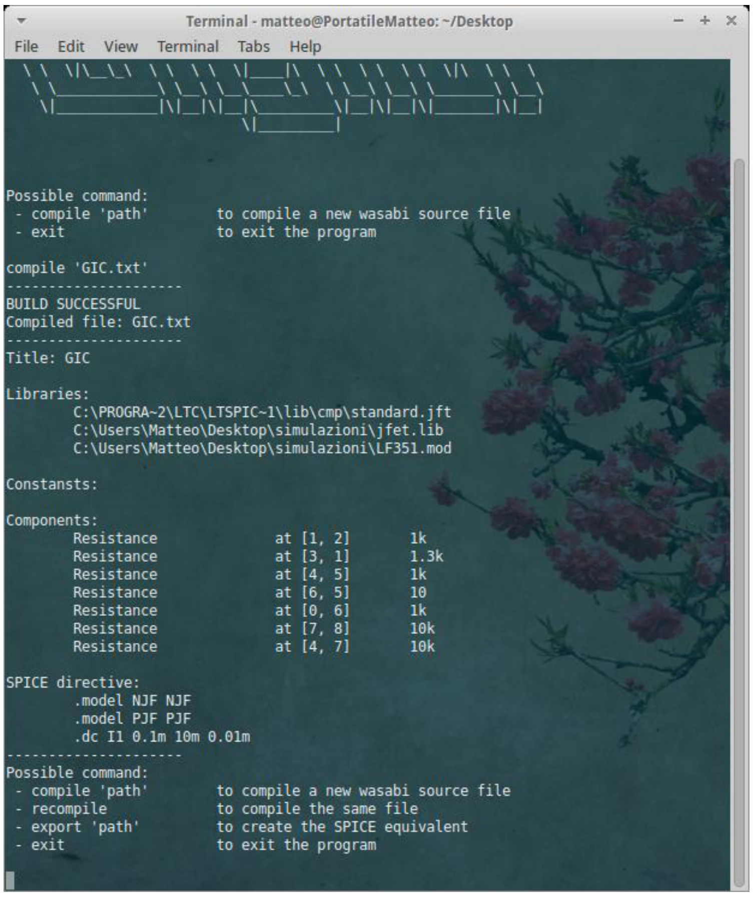


```
matteo@PortatileMatteo:~$ cd Desktop/
matteo@PortatileMatteo:~/Desktop$ java -jar WaSabi.jar

Possible command:
 - compile 'path'       to compile a new wasabi source file
 - exit                 to exit the program
```

As the prompt suggest, the user can compile a file using the command *compile* followed by the path of the file enclosed in apostrophes, or exit the program typing the command *exit*.

A successful compile operation will look like this:

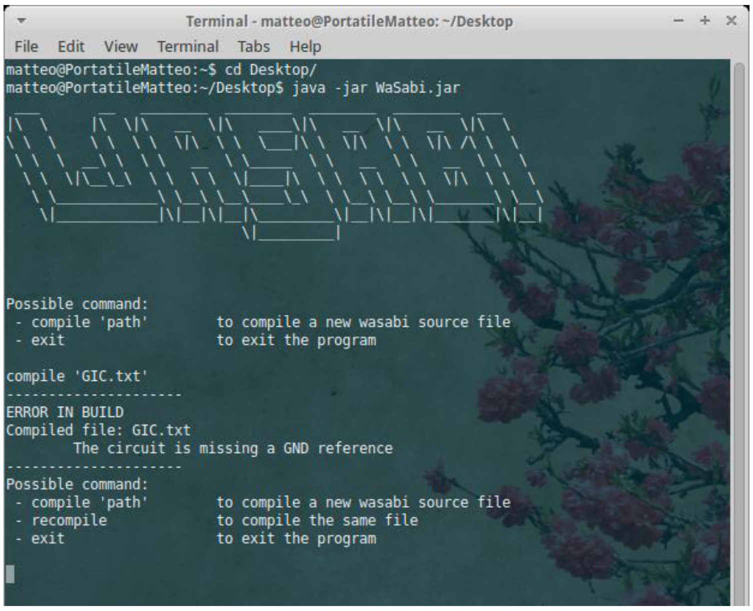
[I]Tested with Java 7.0 and above.



```
  \ \  \|\__\_\  \ \  \ \  \ \  \ \  \|____|\  \ \  \ \  \ \  \ \  \ \  \|\  \ \  \ \  \
   \ \____________\ \__\ \__\____\_\  \ \__\ \__\ \_______\ \__\
    \|____________|\|__|\|__|\_________\|__|\|__|\|_______|\|__|
                            \|_________|



Possible command:
 - compile 'path'        to compile a new wasabi source file
 - exit                  to exit the program

compile 'GIC.txt'
--------------------
BUILD SUCCESSFUL
Compiled file: GIC.txt
--------------------
Title: GIC

Libraries:
        C:\PROGRA~2\LTC\LTSPIC~1\lib\cmp\standard.jft
        C:\Users\Matteo\Desktop\simulazioni\jfet.lib
        C:\Users\Matteo\Desktop\simulazioni\LF351.mod

Constansts:

Components:
        Resistance              at [1, 2]       1k
        Resistance              at [3, 1]       1.3k
        Resistance              at [4, 5]       1k
        Resistance              at [6, 5]       10
        Resistance              at [0, 6]       1k
        Resistance              at [7, 8]       10k
        Resistance              at [4, 7]       10k

SPICE directive:
        .model NJF NJF
        .model PJF PJF
        .dc I1 0.1m 10m 0.01m
--------------------
Possible command:
 - compile 'path'        to compile a new wasabi source file
 - recompile             to compile the same file
 - export 'path'         to create the SPICE equivalent
 - exit                  to exit the program
```

After a successful compilation the tool shows the list of libraries, constants, components and spice directives used in the circuit. From this point, the user can compile another file using the same command as before, recompile the same file using the command *recompile*, or export the resulting SPICE netlist into a file.

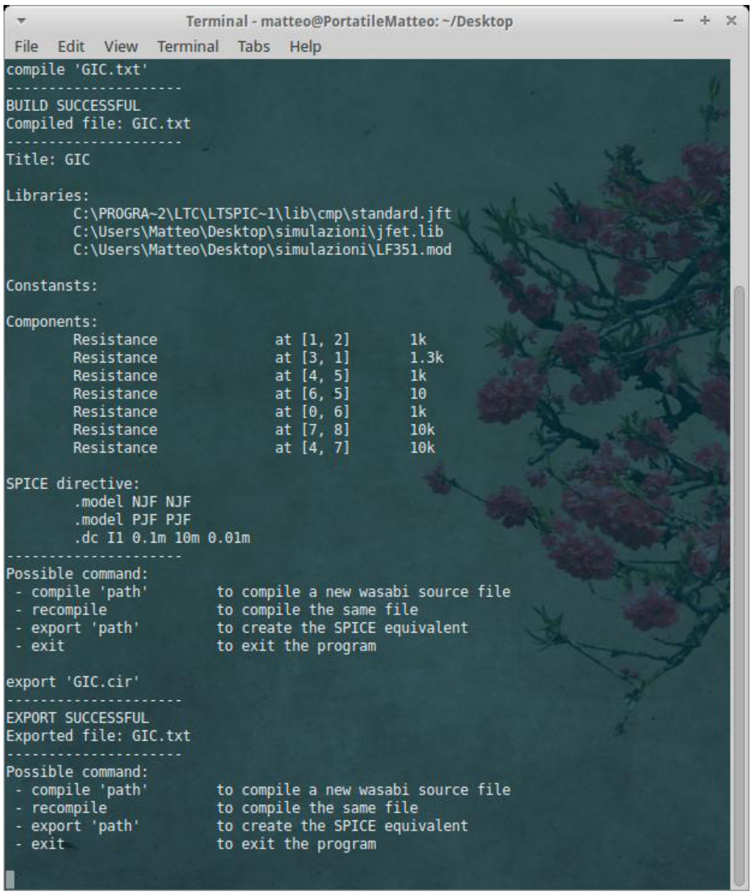
If the compile operation is unsuccessful, the tool will alert the user of eventual errors found in the source, like so:



```
matteo@PortatileMatteo:~$ cd Desktop/
matteo@PortatileMatteo:~/Desktop$ java -jar WaSabi.jar
 ___       ___  ________  ________  ________  ________  ___
|\  \     |\  \|\   __  \|\   ____\|\   __  \|\   __  \|\  \
\ \  \    \ \  \ \  \|\  \ \  \___|\ \  \|\  \ \  \|\ /\ \  \
 \ \  \  __\ \  \ \   __  \ \_____  \ \   __  \ \   __  \ \  \
  \ \  \|\__\_\  \ \  \ \  \|____|\  \ \  \ \  \ \  \|\  \ \  \
   \ \____________\ \__\ \__\____\_\  \ \__\ \__\ \_______\ \__\
    \|____________|\|__|\|__|\_________\|__|\|__|\|_______|\|__|
                            \|_________|



Possible command:
 - compile 'path'        to compile a new wasabi source file
 - exit                  to exit the program

compile 'GIC.txt'
--------------------
ERROR IN BUILD
Compiled file: GIC.txt
        The circuit is missing a GND reference
--------------------
Possible command:
 - compile 'path'        to compile a new wasabi source file
 - recompile             to compile the same file
 - exit                  to exit the program
```

1.3 Exporting to a SPICE file

After a successful compile, the user can choose to export the resulting circuit to a *.cir*, *.net* or *.sp* SPICE file, using the command *export* followed by the path of the file itself; if the file does not exist it will be created, and if the file already exists, the user will be asked for confirmation before overwriting the file.


Terminal - matteo@PortatileMatteo: ~/Desktop
File Edit View Terminal Tabs Help

```
compile 'GIC.txt'
--------------------
BUILD SUCCESSFUL
Compiled file: GIC.txt
--------------------
Title: GIC

Libraries:
        C:\PROGRA~2\LTC\LTSPIC~1\lib\cmp\standard.jft
        C:\Users\Matteo\Desktop\simulazioni\jfet.lib
        C:\Users\Matteo\Desktop\simulazioni\LF351.mod

Constansts:

Components:
        Resistance              at [1, 2]       1k
        Resistance              at [3, 1]       1.3k
        Resistance              at [4, 5]       1k
        Resistance              at [6, 5]       10
        Resistance              at [0, 6]       1k
        Resistance              at [7, 8]       10k
        Resistance              at [4, 7]       10k

SPICE directive:
        .model NJF NJF
        .model PJF PJF
        .dc I1 0.1m 10m 0.01m
--------------------
Possible command:
 - compile 'path'       to compile a new wasabi source file
 - recompile            to compile the same file
 - export 'path'        to create the SPICE equivalent
 - exit                 to exit the program

export 'GIC.cir'
--------------------
EXPORT SUCCESSFUL
Exported file: GIC.txt
--------------------
Possible command:
 - compile 'path'       to compile a new wasabi source file
 - recompile            to compile the same file
 - export 'path'        to create the SPICE equivalent
 - exit                 to exit the program
```

2 Language

2.1 Language introduction

WaSabi is a procedural, non case-sensitive language developed to simplify the creation of SPICE models. It doesen't use a specific character to end an instruction (java uses the ; character for example); it is good practice to separate an instruction from the next one using a new line character, but it's not necessary to ensure a correct execution of the code.

The = character is used as an assignment operator during constants declaration, such as:

```
a=2

b=3.5
```

The language supports the use of comments; a comment is everything enclosed between two sets of −− or everything between a −− and the new line character, such as:

```
--This is a comment--

--And this is also a comment
```

2.2 The program structure

First of all, let us consider overall structure of a WaSabi program:

```
Circuit First_test
        Lib
                --Libraries--
        Const
                --Constant definitions--
        Components
                --List of components and connections--
        Simulate
                --Simulation directives--
```

As can be seen, a program begins with the keyword **Circuit** followed by the title of the project; if the user desires, this initial part can be omitted. The structure is divided into 4 sections, which can be rearranged in any order (meaning that the **Lib** section doesen't need to appear first in the program, and also the other 3 sections can be moved around).

It can be possible to split a section in two parts, like so:

```
Circuit Second_test
        Lib
                --Libraries--
        Components
                --Some components--
        Const
                --Constant definitions--
        Lib
                --More libraries--
        Components
                --Some more components--
        Simulate
                --Simulation directives--
```

The only restriction that applies is on the use of constants: if a constant is used, for example, in the **Component** section, it must have already been defined in a previous **Const** section.

2.3 The libraries section

The libraries section is usually introduced by the **Lib** keyword, but a number of variation of the «libraries» word can be used; the accepted variations are:

- Lib:
- Lib
- Libraries:
- Libraries

This section is used to import libraries in the SPICE model, listing all the absolute or relative paths of the files that must be imported into the SPICE model.

```
--This code imports the "diodes" and "std_bjt" libraries
Lib
        E:\Documents and Settings\SpiceLibraries\diodes.lib
        E:\Documents and Settings\SpiceLibraries\std_bjt.lib

        --This is a relative path, also accepted
        diodes.lib
```

2.4 The constants section

The constants section is usually introduced by the keyword **Const**, but similarly to the libraries section, a number of alternative keywords are permitted:

- Const:
- Const
- Constants:
- Constants

This section can be used to define the constants used in the program. Plese be aware that a constant can be used only after it's been declared in the constants section. Below is an example of the code used in the constants section:

```
Const
        --These are value declarations
        wonderfulVar = 200
        amazingVar = 1p
        v = 4
```

```
--These are voltage sources declarations
vSource = AC[v, pi]
vSource2 = DC[5]

--This is a model declaration
diode = <1N5817>
```

2.4.1 Value declarations

A value declaration is used to assign a number to a constant. In the example above, three value declarations are present: one for the value of 200, stored in a constant called *wonderfulVar*, one for the value of 1 pico, or 10^{-12} (see table 2.1 for the complete list of supported orders of magnitude), stored in *amazingVar*, and a declaration for the variable v, in which the value 4 is saved.

2.4.2 Source declarations

A source declarationis is used to create a definition for a generator and can be for a DC or AC source. This declaration alone cannot define a complete source; the constant defined must be used in the components section to complete the source definition.

The declaration for the DC source uses the following syntax

```
sourceName=DC[value]
```

where the value field represents the voltage or current of the source and can be either a number or a previously declared constant.

The declaration for the AC source uses the following syntax

```
sourceName=AC[module,phase]
```

which declares a general function with the specified module and phase. These sources will be used by spice during the AC analysis; for more informations, please refer to the SPICE manual[I]. Similarly to the DC source declaration, the module and phase arguments can be either a number or a previously defined constant.

2.4.3 Model declarations

A model declaration is used to assign to a variable the name of a specific model: the name of the model must be enclosed in pointy brackets. Note that to use the model the correct library must have been imported in the specific section.

Prefix	*f*	*p*	*n*	*u*	*mill*	*k*	*meg*	*g*	*t*
Value	10^{-15}	10^{-12}	10^{-9}	10^{-6}	10^{-3}	10^{3}	10^{6}	10^{9}	10^{12}

Table 2.1: List of the supported orders of magnitude in a constant declaration.

2.5 The components section

The components section is usually introduced by the keyword **Components**, but as usual, a number of alternative keywords are permitted:

[I]Spice manual - AC analysis

- Comp:
- Comp
- Components:
- Components

This section can be used to declare the components, their values and the interconnections between them. Consider the following example code:

```
Components
                R res at [n4, n3]
                C 3450u at [n5, n4]
                C cap at [n6, n9]
                L 300mill at [n3, Gnd]
                D diode at [n7, GND]
                V vSource at [n3, gnd]
                I DC[1mill] at [n5, n8]
```

As can be seen in the example, the general syntax used to define a component is

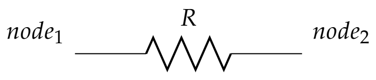
```
typeOfComponent valueOfComponent at [node1,node2,...,nodeN]
```

As usual the value can be either a number or a defined constant.

Notes The nodes names should be alphanumerical strings but some restrictions apply:

- The node *GND* (ground) must exist somewhere in the circuit.
- A component with $node1 = node2$ that is, short circuited on itself, is basically useless therefore not permitted.
- A node must be used at least twice, because using it only once means that a component somewhere in the circuit is connected only to one end.

2.5.1 Resistance



The declaration of a resistance uses the following sintax:

```
R valueOfResistance at [node1,node2]
```

The parameters of the definition are:

$valueOfResistance$ Defines the value of the resistance. Can be either a previously defined constant or a numeric value.

$node1$ Defines the first node to which the resistance is connected.

$node2$ Defines the second node to which the resistance is connected.

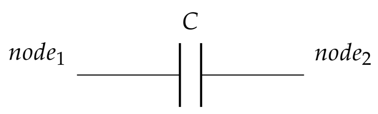
Here are a couple examples of the definition:

```
--This code defines a 200 kOhm resistance between "n1" and "n2"--
Components
        R 200k at [n1,n2]

--Alternative method--
Const
        resValue=200k
Components
        R resValue at [n1,n2]
```

Notes The **R** keyword has alternatives:

- **R**
- **Res**
- **Resistance**

2.5.2 Capacitor



The declaration of a capacitor uses the following sintax:

```
C valueOfCapacitor at [node1,node2]
```

The parameters of the definition are:

valueOfCapacitor Defines the value of the capacitor. Can be either a previously defined constant or a numeric value.

*node*1 Defines the first node to which the capacitor is connected.

*node*2 Defines the second node to which the capacitor is connected.

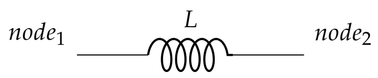
Here are a couple examples of the definition:

```
--This code defines a 7 F capacitor between "n1" and the ground node--
Components
        C 7 at [n1,GND]

--Alternative method--
Const
        capValue=7
Components
        C capValue at [n1,GND]
```

Notes The **C** keyword has alternatives:

- **C**
- **Cap**
- **Capacitance**

2.5.3 Inductor



The declaration of an inductor uses the following sintax:

```
L valueOfInductor at [node1,node2]
```

The parameters of the definition are:

valueOfInductor Defines the value of the inductor. Can be either a previously defined constant or a numeric value.

*node*1 Defines the first node to which the inductor is connected.

*node*2 Defines the second node to which the inductor is connected.

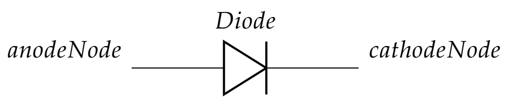
Here are a couple examples of the definition:

```
--This code defines a 30 mH inductor between "n1" and "n2"--
Components
        L 30mill at [n1,n2]

--Alternative method--
Const
        indValue=30mill
Components
        L indValue at [n1,n2]
```

Notes The **L** keyword has alternatives:

- **L**
- **Ind**
- **Inductance**

2.5.4 Diode



The declaration of a diode uses the following sintax:

```
D <modelOfDiode> at [anodeNode,cathodeNode]
```

The parameters of the definition are:

modelOfDiode Defines the model of the diode. Can be either a previously defined constant or string; in both cases the string must be enclosed in pointy brackets.

anodeNode Defines the node to which the anode is connected.

cathodeNode Defines the node to which the cathode is connected.

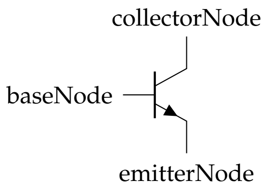
Here are a couple examples of the definition:

```
--This code defines a 1N5817 diode between "n1" and "n2"
--The anode is connected to "n1" and cathode to "n2"

--If the model is not automatically included in your spice installation,
--it must be imported in the lib section

Components
        D <1N5817> at [n1,n2]

--Alternative method--
Const
        diodeModel = <1N5817>
Components
        D diodeModel at [n1,n2]
```

Notes The **D** keyword has alternatives:

- **D**
- **Dio**
- **Diode**

2.5.5 Bipolar Junction Transistor (BJT)



The declaration of a BJT uses the following sintax:

```
B <modelOfBJT> at [collectorNode,baseNode,emitterNode]
```

The parameters of the definition are:

modelOfBJT Defines the model of the BJT. Can be either a previously defined constant or string; in both cases the string must be enclosed in pointy brackets.

collectorNode Defines the node to which the collector is connected.

baseNode Defines the node to which the base is connected.

emitterNode Defines the node to which the emitter is connected.

Here are a couple examples of the definition:

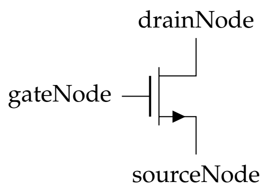
```
--This code defines a LN3391A BJT whose base
--is connected to "n2",collector to "n1" and emitter to "n3"

--If the model is not automatically included in your spice installation,
--it must be imported in the lib section
```

```
Components
        B <LN3391A> at [n1,n2,n3]

--Alternative method--
Const
        bjtModel = <LN3391A>
Components
        B bjtModel at [n1,n2,n3]
```

Notes The **B** keyword has alternatives:

- **B**
- **BJT**

2.5.6 Metal-Oxide-Semiconductor FET (MOSFET)



The declaration of a MOSFET uses the following sintax:

```
M <modelOfMOS> at [drainNode,gateNode,sourceNode]
```

The parameters of the definition are:

modelOfMOS Defines the model of the MOSFET. Can be either a previously defined constant or string; in both cases the string must be enclosed in pointy brackets.

drainNode Defines the node to which the drain is connected.

gateNode Defines the node to which the gate is connected.

sourceNode Defines the node to which the source is connected.

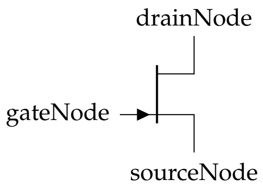
Here are a couple examples of the definition:

```
--This code defines a FDS6912 MOSFET whose
--gate is connected to "n2", drain to "n1" and source to "n3"

--If the model is not automatically included in your spice installation,
--it must be imported in the lib section

Components
        M <FDS6912> at [n1,n2,n3]

--Alternative method--
Const
        mosModel = <FDS6912>
Components
        M mosModel at [n1,n2,n3]
```

Notes The **M** keyword has alternatives:

- **M**
- **MOS**
- **MOSFET**

2.5.7 Junction FET



The declaration of a JFET uses the following sintax:

```
J <modelOfJFET> at [drainNode,gateNode,sourceNode]
```

The parameters of the definition are:

modelOfJFET Defines the model of the JFET. Can be either a previously defined constant or string; in both cases the string must be enclosed in pointy brackets.

drainNode Defines the node to which the drain is connected.

gateNode Defines the node to which the gate is connected.

sourceNode Defines the node to which the source is connected.

Here are a couple examples of the definition:

```
--This code defines a 2N4117 JFET whose
--gate is connected to "n2", drain to "n1" and source to "n3"

--If the model is not automatically included in your spice installation,
--it must be imported in the lib section

Components
        J <2N4117> at [n1,n2,n3]

--Alternative method--
Const
        jfetModel = <2N4117>
Components
        J jfetModel at [n1,n2,n3]
```

Notes The **J** keyword has alternatives:

- **J**
- **JFET**

2.5.8 Generic model

The declaration of a generic model uses the following sintax:

```
Model <specificModel> at [node1,node2,...,nodeN]
```

The parameters of the definition are:

specificModel Defines the model of the component. Can be either a previously defined constant or string; in both cases the string must be enclosed in pointy brackets.

*node*1...*nodeN* Defines the nodes to which the component is connected

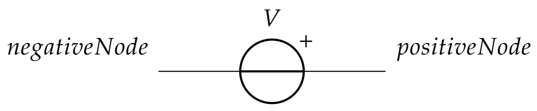
Here are a couple examples of the definition:

```
--This code defines a LT1001 low noise amplifier--
--The positive input is connected to the node "inP", the negative to "inN"

--The positive supply pin is connected to the "splyP" node,
--the negative to "splyN"

--The output pin is connected to the "out" node

--If the model is not automatically included in your spice installation,
--it must be imported in the lib section--

Components
        Model <LT1001> at [inP,inN,splyP,splyN,out]

--Alternative method--
Const
        genModel = <LT1001>
Components
        Model genModel at [inP,inN,splyP,splyN,out]
```

Notes The **Model** keyword has alternatives:

- **Model**
- **Mod**

2.5.9 Voltage source



The declaration of a voltage source uses the following sintax:

```
V sourceDefinition at [positiveNode,negativeNode]
```

The parameters of the definition are:

sourceDefinition Defines the source using a constant as presented in section 2.4, or using a string containing the direct definition of the source.

positiveNode Defines the node to which the positive pole of the source is connected.

negativeNode Defines the node to which the negative pole of the source is connected.

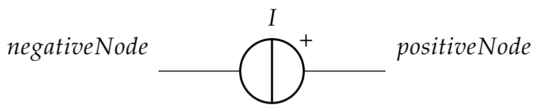
Here are a couple examples of the definition:

```
--This code defines a 5V DC source between GND and n1--
Const
        sourceDefinition = DC[5]
Components
        V sourceDefinition at [n1,GND]


--Alternative method, more direct--
Components
        V DC[5] at [n1,GND]
```

Notes The **V** keyword has alternatives:

- **V**
- **Vol**
- **Voltage**

2.5.10 Current source



The declaration of a voltage source uses the following sintax:

```
I sourceDefinition at [positiveNode,negativeNode]
```

The parameters of the definition are:

sourceDefinition Defines the source using a constant as presented in section 2.4, or using a string containing the direct definition of the source.

positiveNode Defines the node to which the positive pole of the source is connected.

negativeNode Defines the node to which the negative pole of the source is connected.

Here are a couple examples of the definition:

```
--This code defines a 3A DC source between GND and n1--
Const
        sourceDefinition = DC[3]
Components
        I sourceDefinition at [n1,GND]


--Alternative method, more direct--
Components
        I DC[3] at [n1,GND]
```

Notes The **I** keyword has alternatives:

- **I**
- **Cur**
- **Current**

2.6 The simulation directives section

The directives section is introduced using the keyword **Sim**, but the following alternatives are permitted:

- Sim:
- Sim
- Simulate:
- Simulate

This section can be used to include SPICE simulation directives in order to analyze the circuit. More informations about the accepted directives can be found in the SPICE manual[II]. Here is an example of the usage of SPICE directives:

```
--This code tells spice to find the operation point and to do an AC sweep,
--from 1 Hz to 10kHz, analyzing 10 points per decade
Sim
        .op
        .ac DEC 10 1 10K
```

2.7 List of keywords

The following table contains all the keywords used by the wasabi language: the user should not use them as variable names to avoid syntactical errors.

[II] http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/UserGuide/analyses_fr.html

Keyword	Description	Keyword	Description
at	Used to indicate the nodes to which the component is connected. See section 2.5.	*JFET*	Used in the declaration of a JFET. See section 2.5.7.
AC	Used in the declaration of an AC source. See section 2.4.2.	*k*	Used to indicate the SI prefix of kilo, 10^3.
B	Used in the declaration of a BJT. See section 2.5.5.	*L*	Used in the declaration of an inductance. See section 2.5.3.
BJT	Used in the declaration of a BJT. See section 2.5.5.	*Lib*	Used to introduce the libraries section. See section 2.3.
C	Used in the declaration of a capacitor. See section 2.5.2.	*Libraries*	Used to introduce the libraries section. See section 2.3.
Cap	Used in the declaration of a capacitor. See section 2.5.2.	*M*	Used in the declaration of a MOSFET. See section 2.5.6.
Capacitance	Used in the declaration of a capacitor. See section 2.5.2.	*meg*	Used to indicate the SI prefix of mega, 10^6.
Circuit	Used to introduce the title.	*mill*	Used to indicate the SI prefix of milli, 10^{-3}.
Comp	Used to introduce the components section. See section 2.5.	*mod*	Used in the declaration of a generic model. See section 2.5.8.
Components	Used to introduce the components section. See section 2.5.	*model*	Used in the declaration of a generic model. See section 2.5.8.
Const	Used to introduce the constants section. See section 2.4.	*MOS*	Used in the declaration of a MOSFET. See section 2.5.6.
Constants	Used to introduce the constants section. See section 2.4.	*MOSFET*	Used in the declaration of a MOSFET. See section 2.5.6.

Keyword	Description	Keyword	Description
Cur	Used in the declaration of a current source. See section 2.5.10.	*n*	Used to indicate the SI prefix of nano, 10^{-9}.
Current	Used in the declaration of a current source. See section 2.5.10.	*p*	Used to indicate the SI prefix of pico, 10^{-12}.
D	Used in the declaration of a diode. See section 2.5.4.	*pi*	Used to indicate the number π[III].
DC	Used in the declaration of a DC source. See section 2.4.2.	*R*	Used in the declaration of a resistance. See section 2.5.1.
Dio	Used in the declaration of a diode. See section 2.5.4.	*Res*	Used in the declaration of a resistance. See section 2.5.1.
Diode	Used in the declaration of a diode. See section 2.5.4.	*Resistance*	Used in the declaration of a resistance. See section 2.5.1.
f	Used to indicate the SI prefix of femto, 10^{-15}.	*Sim*	Used to introduce the directives section. See section 2.6.
g	Used to indicate the SI prefix of giga, 10^{9}.	*Simulate*	Used to introduce the directives section. See section 2.6.
GND	Used to indicate the ground reference. Must be present at least one time in all the circuit.	*t*	Used to indicate the SI prefix of tera, 10^{12}.
I	Used in the declaration of a current source. See section 2.5.10.	*u*	Used to indicate the SI prefix of micro, 10^{-6}.
Ind	Used in the declaration of an inductance. See section 2.5.3.	*V*	Used in the declaration of a voltage source. See section 2.5.9.
Inductance	Used in the declaration of an inductance. See section 2.5.3.	*Vol*	Used in the declaration of a voltage source. See section 2.5.9.
J	Used in the declaration of a JFET. See section 2.5.7.	*Voltage*	Used in the declaration of a voltage source. See section 2.5.9.

2.8 Semantic errors

This section is dedicated to all the possible semantic errors that the user can encounter in his use of the WaSabi language.

2.8.1 Constants with the same name

Description This error occours when there are two or more constants defined with the same name and it is probably caused by the fact that the user forgot he had already defined the constant he intended to use. It is identified by an error message that shows: «Constant amazingConstant at line 10 is already defined.».

How to fix To fix this error the user needs to rename the constant declared in the line that generated the error itself (line 10 in the previous example); beware that this operation may break other lines where the costant was used.

2.8.2 Constant not defined

Description This error occours when a undefined constant is used, and it is probably caused by the user forgetting to declare a constant or misstyping the latter in its definition or usage. It is identified by an error message that shows: «The constant wonderfulConstant used in line 18 isn't defined.».

How to fix To fix this error the user needs to declare the constant with the correct name in the specific section.

2.8.3 Duplicate library

Description This error occours when the same library is imported twice. It is identified by an error message that shows: «The library C:\libraries folder\very important library.mod at line 5 is already imported.».

How to fix To fix this error the user needs delete the line that generated the error or, in case it was a typo, fix the name/path of the imported library.

2.8.4 Floating node

Description This error occours when a node is only used once, meaning that a component has only one end connected to the circuit, and it is usually caused by the user misstyping the node name. It is identified by an error message that shows: «The node 'bestNodeEver' is only used once, it should be used at least twice.».

How to fix To fix this error the user needs to rename the node that generated the error, in order to connect its component to the rest of the circuit.

2.8.5 No components in the circuit

Description This error occours when the circuit has no components. It is identified by an error message that shows: «The circuit must have at least a component.».

How to fix To fix this error the user needs to add components to the circuit.

2.8.6 No GND reference

Description This error occours when the circuit has no GND node, it is probably caused by the user forgetting to add one or misstyping the only one present. It is identified by an error message that shows: «The circuit is missing a GND reference».

How to fix To fix this error the user needs to add a GND node to the circuit.

2.8.7 Parallel voltage exception

Description This error occours when two voltage sources are connected in parallel. It is identified by an error message that shows: «The generator at line 29 is in parallel with another voltage generator».

How to fix To fix this error the user must move or delete one of the two sources, so that they are no longer connected in parallel.

2.8.8 Short circuits

Description This error occours when a component is connected between the same node. It is identified by an error message that shows: «The bipole at line 14 is ignored because is short circuited».

How to fix To fix this error the user delete the short circuited component or connect one of its nodes somewhere else.

2.8.9 Using the wrong constants

Description This error occours when constant is used in the wrong way: for example using a model as a value, or a value as a waveform. It is identified by an error message that shows: «The constant 'totallyNotaValue' at line 14 is a waveform but is used as a value».

How to fix To fix this error the user must declare a constant of the correct type and use it in the line indicated by the error message.

2.8.10 Declaring components with value at zero

Description This error occours when a component that requires a value, such as a resistance, capacitor or inductor, is declared with that value being zero. It is identified by an error message that shows: «The value for the basic component at line 14 cannot be zero.».

How to fix To fix this error the user must declare the component using a non-zero value or, given that a $0\,\Omega$ resistance and a $0\,H$ inductor are equivalent to a short circuit and a $0\,F$ capacitor is equivalent to an open circuit, modify the circuit to reflect these choices.

3 Examples

BJT amplifier

The first example will be a BJT amplifier such as the one represented in figure 3.1.

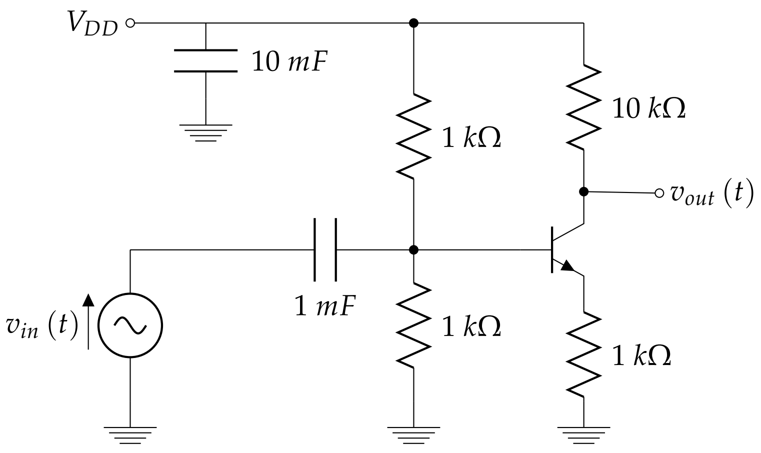


Figure 3.1: A BJT amplifier

Wasabi code

```
Circuit Amplifier

Lib
        C:\PROGRA~2\LTC\LTSPIC~1\lib\cmp\standard.bjt

Components
        R 10k at [VDD,n3]
        R 1k at [n4,gnd]
        R 1k at [VDD,n2]
        R 1k at [n2,gnd]

        C 1mill at [n2,n1]
        C 10mill at [VDD,gnd]

        V DC[15] at [VDD, gnd]
        V AC[1mill,0] at [n1,gnd]

        B <NPN> at[n3,n2,n4]

Simulate
        .model NPN NPN
        .model PNP PNP
        .ac dec 100 10 1000
```

Operational amplifier in astable configuration

This example will be about an operational amplifier (model LT1001) connected in astable configuration; the complete circuit is shown in figure 3.2.

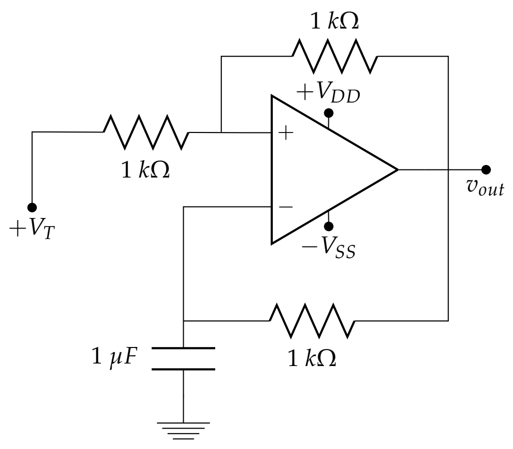


Figure 3.2: Operational amplifier in astable configuration

Wasabi code

```
Circuit Astable

Lib:
        LTC.lib

CONST
        res = 1k

Simulate
        .tran 0.1

Components
        C 1u at [n1,Gnd]
        V DC[10] at [p1,Gnd]
        V DC[10] at [Gnd,p2]
        Model <LT1001> at[n3,n1,p1,p2,n2]
        R val at [n2,n3]
        R val at [n3,Gnd]
        R val at [n2,n1]
```

General immittance converter

The next example is a general immittance converter; the complete circuit is shown in figure 3.3.

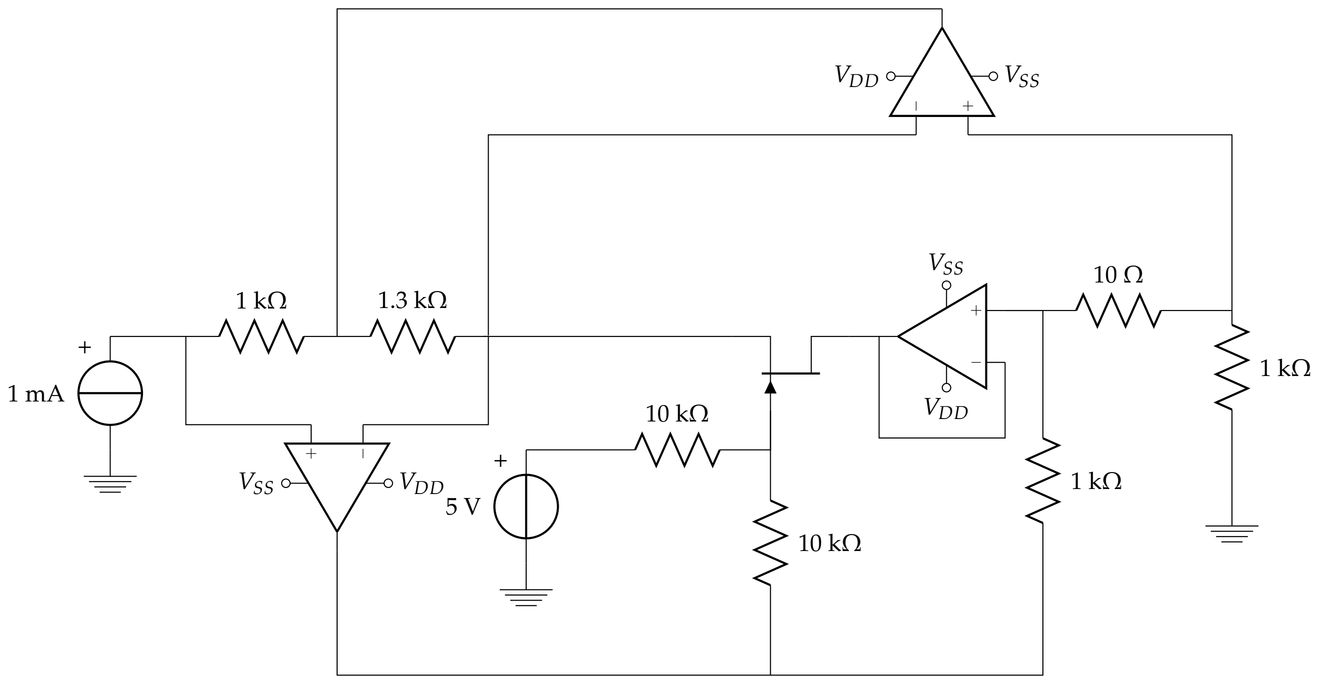


Figure 3.3: General immittance converter

Wasabi code

```
Circuit GIC

Lib
        C:\PROGRA~2\LTC\LTSPIC~1\lib\cmp\standard.jft
        C:\Users\Matteo\Desktop\simulazioni\jfet.lib
        C:\Users\Matteo\Desktop\simulazioni\LF351.mod

Components
        R 1000 at [n1,vx]
        R 1300 at [vs,n1]
        R 1k at [n5,n3]
        R 10 at [n2,n3]
        R 1k at [gnd,n2]
        R 10k at [n4,vc]
        R 10k at [n5,n4]

        J <J105> at [vd,n4,vs]

        V DC[26] at [vdd,gnd]
        V DC[26] at [gnd,vss]
        V DC[5] at [vc,gnd]

        I DC[100u] at [gnd,vx]

        Model <LF351NS> at [n3,vd,vdd,vss,vd]
        Model <LF351NS> at [vx,vs,vdd,vss,n5]
        Model <LF351NS> at [n2,vs,vdd,vss,n1]

Simulate
```

```
.model NJF NJF
.model PJF PJF
.dc I1 0.1m 10m 0.01m
```

Relay control circuit

Another example the control circuit of a relay (represented with an inductance) using a NE555 integrated circuit in astable configuration; the complete circuit is shown in figure 3.4.

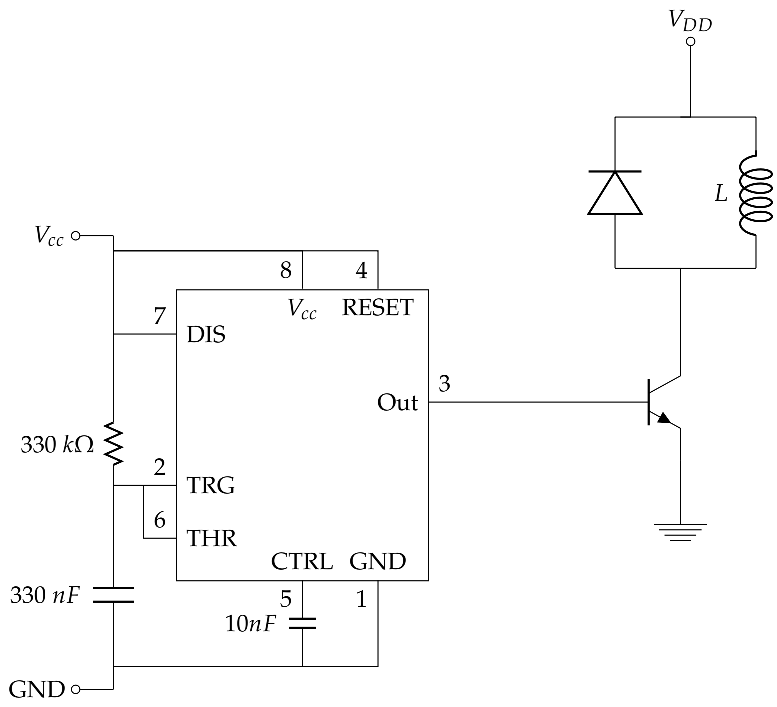


Figure 3.4: Relay control circuit with flyback diode.

Wasabi code

```
Circuit Controlled_Relay

Lib
        C:\Program Files (x86)\LTC\LTspiceIV\lib\cmp\standard.dio
        C:\Program Files (x86)\LTC\LTspiceIV\lib\cmp\standard.bjt
        NE555.sub

Components
        L 10u at [n1,n3]

        D <1N4148> at [n3,n1]

        B <2N2222> at [n3,out,gnd]

        R 150k at[n2,dis]
        R 332k at[dis,trig]
        V DC[5] at [n2,gnd]
        V DC[15] at [n1,gnd]

        C 100n at [n4,gnd]
        C 1u at [trig,gnd]

        model <NE555> at [gnd,trig,out,n2,n4,trig,dis,n2]

Simulate
        .model D D
        .model NPN NPN
        .model PNP PNP

        .tran 5 startup
```

Internal circuit of a MOSFET differential amplifier

The last example is a simplified version of the internal circuit of a BJT-based operational amplifier, whose circuit is shown in figure 3.5.

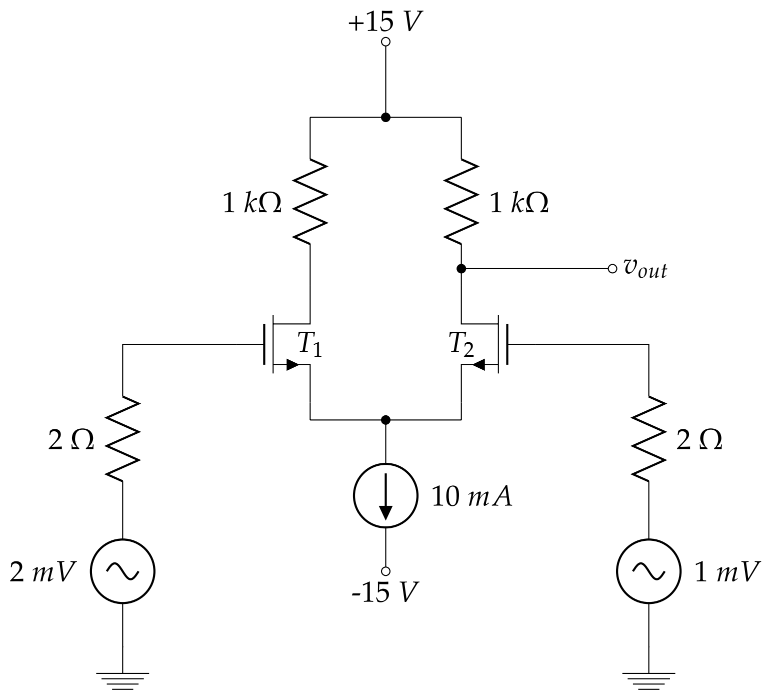


Figure 3.5: Internal circuit of a BJT operational amplifier.

Wasabi code

```
circuit MOSFETdifferential

Lib
        C:\PROGRA~2\LTC\LTSPIC~1\lib\cmp\standard.mos

Components
```

```
        R 1k at [VDD,n1]
        R 1k at [VDD,VOUT]
        R 2 at [VP,n3]
        R 2 at [VL,n4]

        V DC[15] at [VDD,gnd]
        V DC[15] at [VSS,gnd]
        V AC[2mill,0] at [n3,gnd]
        V AC[1mill,0] at [n4,gnd]

        I DC[10mill] at [n2,VSS]

        model <NMOS> at [n1,VP,n2,n2]
        model <NMOS> at [VOUT,VL,n2,n2]

Simulate
        .model NMOS NMOS
        .model PMOS PMOS
        .op
        .ac dec 100 100 10000
```