T-Spice 12 User Guide—Contents

(Continued)

T-Spice 12 User Guide—Contents

(Continued)

T-Spice 12 User Guide—Contents

(Continued)

1 Introduction

Documentation Conventions

This section contains information about the typographical and stylistic conventions used in this user guide.

Special Fonts

The following inline references are represented by a bold font:

- Menu and simulation commands (For example: **.print tran v(out)**.)
- Literal user input (For example: Enter **14.5**.)
- Program output (For example: S-Edit generates names for the ports on the symbol based on the **PAD** string.)
- All dialog elements—fields, checkboxes, drop-down menus, titles, etc. (For example: Click **Add**.)

Freestanding quotations of input examples, file listings, and output messages are represented by a constant-width font—for example:

```
.ac DEC 5 1MEG 100MEG
```

Variables for which context-specific substitutions should be made are represented by bold italics—for example, ***myfile*.tdb**.

Sequential steps in a tutorial are set off with a checkbox (☑) in the margin.

References to mouse buttons are given in all capitals—for example, MOVE/EDIT. When a key is to be pressed and held while a mouse button is used, the key and button are adjoined by a plus sign (+). For example, **Shift**+SELECT means that the **Shift** key is pressed and held while the SELECT mouse button is used.

The terms “left-click,” “right-click,” and “middle-click” all assume default mappings for mouse buttons.

Text omitted for clarity or brevity is indicated by an ellipsis (**…**).

Menu Commands and Dialog Titles

Elements in hierarchical menu paths are separated by a **>** sign. For example, **File > Open** means the **Open** command in the **File** menu.

Tabs in dialog boxes are set off from the command name or dialog box title by a dash. For example, **Setup > Layers—General** and **Setup Layers—General** both refer to the **General** tab of the **Setup Layers** dialog.

Special Keys

Special keys are represented by the following abbreviations:

Key	*Abbreviation*
Shift	**Shift**
Enter	**Enter**
Control	**Ctrl**
Alternate	**Alt**
Backspace	**Back**
Delete	**Del**
Escape	**Esc**
Insert	**Ins**
Tab	**Tab**
Home	**Home**
End	**End**
Page Up	**PgUp**
Page Down	**PgDn**
Function Keys	**F1 F2 F3** …
Arrow Keys	↓, ←, →, ↑

When certain keys are to be pressed simultaneously, their abbreviations are adjoined by a plus sign (+). For example, **Ctrl**+**R** means that the **Ctrl** and **R** keys are pressed at the same time.

When certain keys are to be pressed in sequence, their abbreviations are separated by a space ( ). For example, **Alt**+**E R** means that the **Alt** and **E** keys are pressed at the same time and then released, immediately after which the **R** key is pressed.

Abbreviations for alternative key-presses are separated by a slash (/). For example, **Shift**+↑ **/** ↓ means that the **Shift** key can be pressed together with either the up (↑) arrow key or the down (↓) arrow key.

2 Getting Started

Introduction

This chapter describes the T-Spice user interface and provides a simple tutorial on the basics of T-Spice.

User Interface

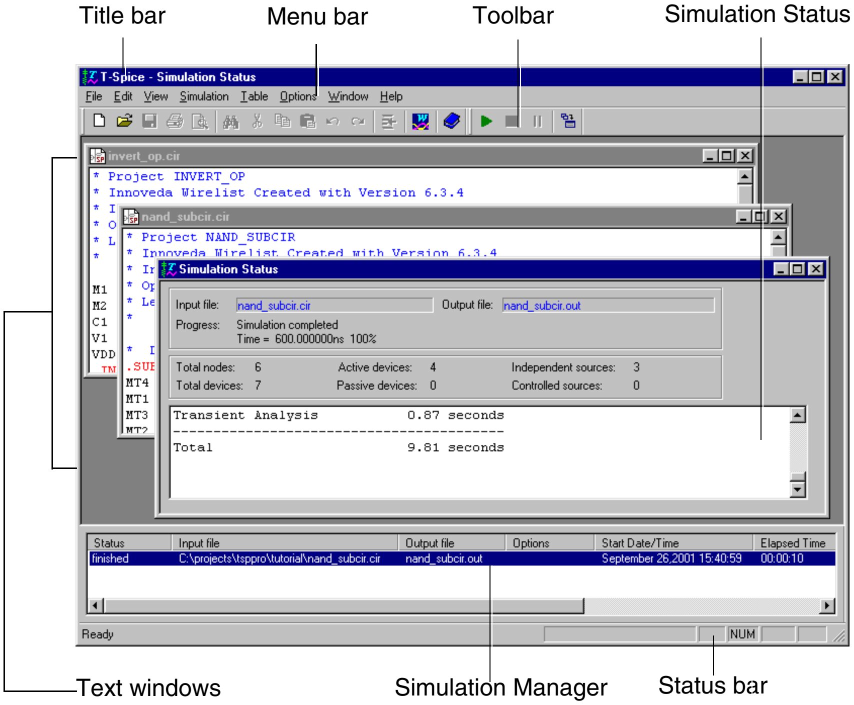
The T-Spice user interface consists of the following elements:

- Title bar
- Menu bar
- Toolbars
- Status bar
- Simulation Manager
- Simulation Status window

Commands on the **View** menu toggle the display of the last four items.

In addition, text windows display the contents of input and output files.

The T-Spice user interface looks like this:



```
* Project INVERT_OP
* Innoveda Wirelist Created with Version 6.3.4
```

```
* Project NAND_SUBCIR
* Innoveda Wirelist Created with Version 6.3.4
```

Input file: nand_subcir.cir
Output file: nand_subcir.out
Progress: Simulation completed
Time = 600.000000ns 100%

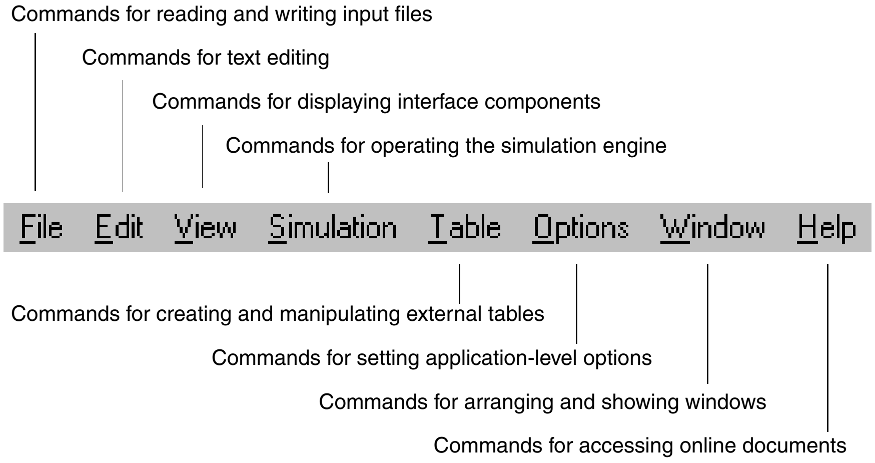
Total nodes: 6
Total devices: 7
Active devices: 4
Passive devices: 0
Independent sources: 3
Controlled sources: 0

```
Transient Analysis            0.87 seconds
----------------------------------------
Total                         9.81 seconds
```

Status	Input file	Output file	Options	Start Date/Time	Elapsed Time
finished	C:\projects\tsppro\tutorial\nand_subcir.cir	nand_subcir.out		September 26,2001 15:40:59	00:00:10

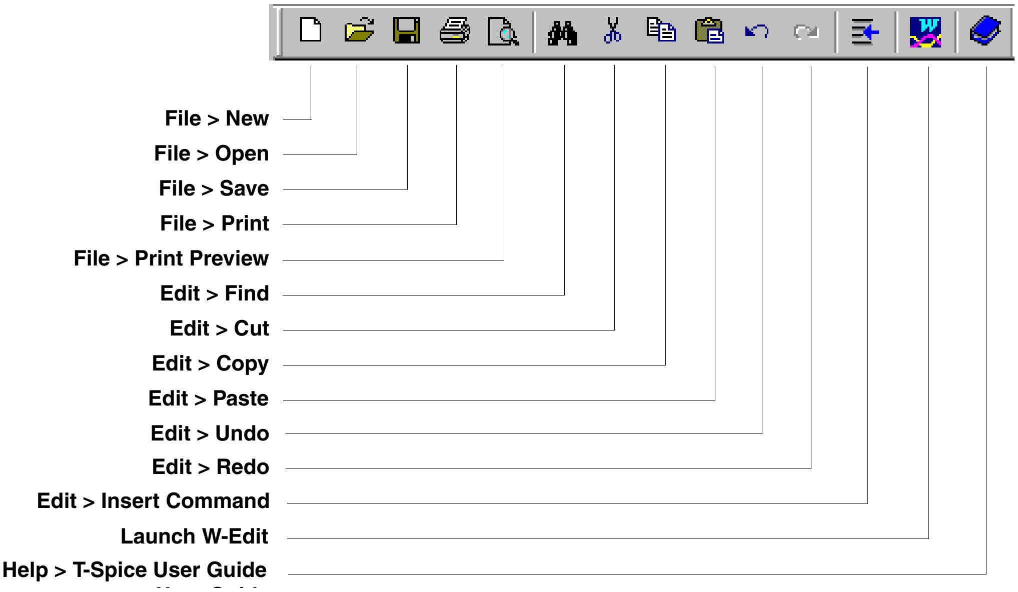
Menu Bar

The menu bar contains the names of the T-Spice command menus. The **Edit** and **Window** menus are only available when the active window contains a text file.

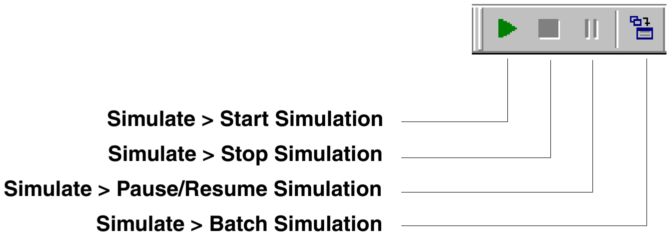

Commands for reading and writing input files
Commands for text editing
Commands for displaying interface components
Commands for operating the simulation engine
File
Edit
View
Simulation
Table
Options
Window
Help
Commands for creating and manipulating external tables
Commands for setting application-level options
Commands for arranging and showing windows
Commands for accessing online documents

Toolbars

The Command toolbar contains buttons with icons representing the most commonly used menu commands.

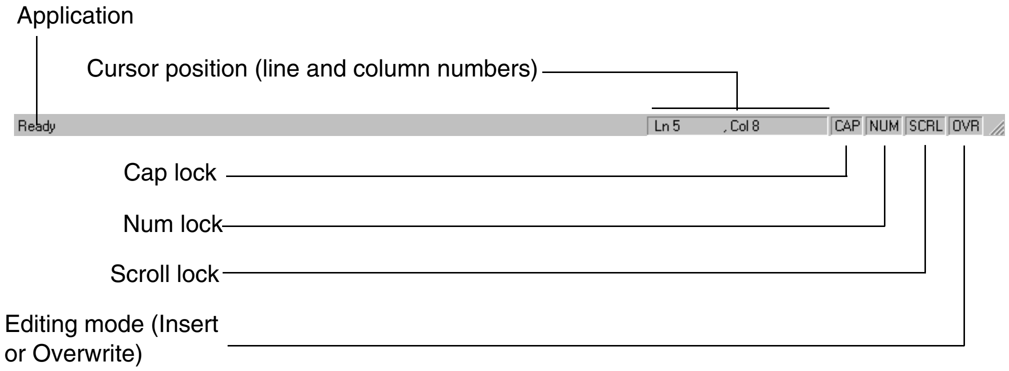


The Simulation toolbar contains buttons with icons representing simulation commands.


Simulate > Start Simulation
Simulate > Stop Simulation
Simulate > Pause/Resume Simulation
Simulate > Batch Simulation

Status Bar

When the pointer is positioned over a button in the toolbar, the left side of the status bar contains a short description of the button’s function. When a text window in the display area is active, the status bar provides information useful for editing.


Application
Cursor position (line and column numbers)
Ready
Ln 5
, Col 8
CAP
NUM
SCRL
OVR
Cap lock
Num lock
Scroll lock
Editing mode (Insert or Overwrite)

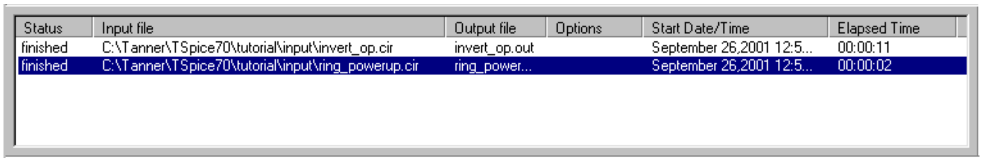
Display Area

The *display area* consists of the entire T-Spice window not occupied by the title, menu bar, toolbars, Simulation Manager, Simulation Status window, or status bar. In this area, input files are viewed and edited and simulation status information is displayed.

Simulation Manager

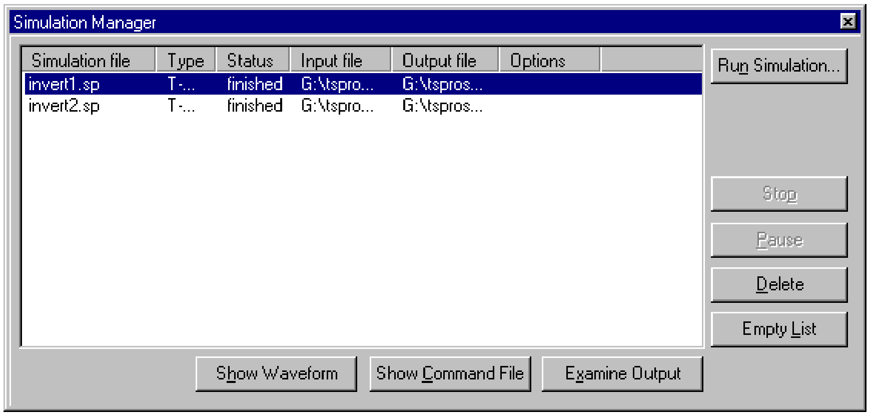
The Simulation Manager allows you to control and monitor all T-Spice simulations. Use **View > Simulation Manager** to display or hide the Simulation Manager. You can dock the Simulation Manager against any edge of the application window or allow it to float within the application window.

When the Simulation Manager is docked against the upper or lower edge of the application window, it will look like this:



Status	Input file	Output file	Options	Start Date/Time	Elapsed Time
finished	C:\Tanner\TSpice70\tutorial\input\invert_op.cir	invert_op.out		September 26,2001 12:5...	00:00:11
finished	C:\Tanner\TSpice70\tutorial\input\ring_powerup.cir	ring_power...		September 26,2001 12:5...	00:00:02

When the Simulation Manager is undocked, it will look like this:



Simulation Manager

Simulation file	Type	Status	Input file	Output file	Options
invert1.sp	T-...	finished	G:\tspro...	G:\tspros...	
invert2.sp	T-...	finished	G:\tspro...	G:\tspros...	

You can dock or undock the Simulation Manager using one of several techniques:

- Dragging it to the desired location
- Selecting **Docking view** from the pop-up menu. (Right-click in any field to access this menu.)
- When undocked, by clicking the title bar
- When docked, by double-clicking any edge of the dialog box

You can use the Simulation Manager to monitor multiple simulations at one time. By selecting a simulation entry in the Simulation Manager, you can switch to different simulations. To view the output for a simulation, select it in the Simulation Manager. The content in the Simulation Status window is based on the current simulation.

Each simulation occupies one row, and each row has six attributes:

Status

Simulation status. Possible states include:

- **Queued**—the simulation is in the queue and will run when the simulation engine is available.
- **Running**—the simulation is underway.
- **Paused**—simulation has been suspended by the user.
- **Finished**—the simulation has run and output is available.
- **Stopped**—the simulation was stopped by the user.
- **Failed**—the simulation has failed to run.

Input file

Full pathname of the input file

Output file

Full pathname of the output file

Options

Lists all command-line options entered in the **Command Options** field of the **Run Simulation** dialog. For further information, see **Simulation > Run Simulation** (page 511) and Command-Line Options on page 28.

Start Date/Time

The date and time at which the simulation began execution.

Elapsed Time

The total simulation run time in **hh:mm:ss** format. Pausing a simulation will not interrupt the measurement of elapsed time.

When the Simulation Manager is *undocked*, the following buttons are available to control queued simulations:

Run Simulation

Invokes **Simulation > Run Simulation** (page 511). If a file is highlighted, the appropriate file and path names will automatically populate the **Input file** and **Output file** fields.

Stop

Stops simulation processing for the highlighted file. Once a simulation is stopped, it cannot be resumed.

Pause/Resume

Pauses simulation when the status of the highlighted file is **Running**, and resumes simulation when the status of the highlighted file is **Paused**.

Delete

Removes the highlighted file from the simulation queue. If the simulation is running or paused, you will be prompted to stop processing on the file before it is deleted.

Empty List	Removes all files from the simulation queue. If any simulations are running or paused, you will be prompted to stop processing before they are deleted.
Show Waveform	Invokes the W-Edit waveform viewer for the selected simulation.
Show Netlist	Opens a window with the selected input file. If the selected file is already open, makes that window active.
Show Output	Opens a window with the output file (**.out**) corresponding to the selected simulation. If already open, makes that window active.

Right-clicking in any field of this window opens a pop-up menu with the options **Run Simulation**, **Delete**, **Empty List**, **Show Waveform**, **Show Netlist**, and **Show Output**, as described above. Two additional menu items specify whether the window will be visible (**Hide**) and whether it will be docked (**Docking view**).

Note: The simulation-specific options **Delete**, **Show Waveform**, **Show Netlist**, and **Show Output** are only enabled when you right-click on a simulation entry.

Simulation Status

The Simulation Status window displays output messages from the T-Spice simulation engine. There is only one Simulation Status window, and all simulations, whether local or remote, display their output in this window. When you select a simulation in the Simulation Manager, the Simulation Status window displays the results for that simulation.

The **Simulation Status** window shows simulation statistics and progress information, as well as any warnings or error messages, for the currently running or most recently completed simulation

Input file	Name of the input file.
Output file	Name of the output file.
Progress	The type of simulation, duration in nanoseconds, and percentage of the simulation completed.
Total nodes	Total number of nodes simulated.
Total devices	Total number of devices simulated.
Active devices	Total number of active devices simulated.
Passive devices	Total number of passive devices simulated.
Independent sources	Total number of independent sources simulated.
Controlled sources	Total number of controlled sources simulated.

Input Files

At the heart of T-Spice's operation is the *input file* (also known as the *circuit description*, the *netlist*, or the *input deck*). This is a plain text file that contains the *device statements* and *simulation commands*, drawn from the SPICE *circuit description language*, with which T-Spice constructs a model of the circuit to be simulated. Input files can be created and modified with any text editor, though the text editor integrated with T-Spice is ideal as it includes default and fully-customized syntax highlighting.

Input files can be very long and complex, but they do not have to be written from scratch; they can be efficiently created using the export facility of a schematic editor (such as S-Edit™), or the extraction facility of a layout editor (such as L-Edit™). In addition, T-Spice includes a **Command Tool** (page 17) that automates error-free SPICE language entry.

Any number of text files can be open at once, each in its own window in the display area. However, only one window can be "active" at any given time, and only an input file displayed in an *active* window can be edited and simulated.

File Synchronization

T-Spice has file synchronization features that allow you to control how files are saved and updated.

For input files, you can set an application-level default (auto-load) so that text files open within T-Spice are automatically reloaded if they are modified outside of T-Spice. You can also set a default for text files that have been modified inside T-Spice so that prior to being used they are automatically saved (or not, or with a prompt). Similarly, you can set a gobal default for output files so that they will always be overwritten without a prompt (see Setup Options on page 24).

If the file synchronization feature poses an inconvenience due to a network configuration or other issues, you can disable it by starting T-Spice with the **-y** command-line flag (see Command-Line Options on page 28).

Editing Text Files

When the pointer is in an active text window, it becomes an I-beam. The position in the input file at which text is to be added is marked by a *blinking cursor*, and the status bar displays the cursor position, editing mode, and other information as described in Status Bar on page 9. You can toggle between the two editing modes by pressing the **Ins** key. In Insert mode, text is added *between* the characters separated by the blinking cursor. In Overwrite mode, text is added *in place* of the character to the right of the blinking cursor.

The cursor can be moved by clicking the pointer at the desired location. Sections of text can be selected by clicking and dragging the pointer. Double-clicking selects a word.

Alternatively, all these functions are accessible from the keyboard, as follows.

Keys	***Functions***
↑ ↓ ← →	Move the cursor in the indicated direction
Home / **End**	Move the cursor to the beginning or end of the line
PgUp / **PgDn**	Move the cursor up or down one page (scrolling the window with it)
Shift + ↑ ↓ ← →	Extend the selection in the indicated direction
Shift + **Home** / **End**	Extend the selection to the beginning or end of the line
Shift + **PgUp** / **PgDn**	Extend the selection up or down one page
Shift + **Ctrl** + **Home** / **End**	Extend the selection to the beginning or end of the file

Selections of text can be manipulated with the **Cut**, **Copy**, **Paste**, and **Clear** commands in the **Edit** menu (or, in the case of the first three, by clicking ✂, ⎘, and 📋 in the toolbar). The **Cut** and **Copy** commands put deleted or duplicated text onto the clipboard, and from there the text can be placed elsewhere with the **Paste** command. The **Clear** command simply deletes text without adding it to the Clipboard.

Delimiters

T-Spice allows several different characters to be used as comment delimiters, including the asterisk (*****), dollar sign (**$**), semicolon (;), and C-language style slash (**/*** or ***/**). However, the T-Spice text editor will only color-code comment text when:

- An asterisk (*****) used as delimiter is placed in the first column of the text editor
- A dollar sign (**$**) is used as delimiter in any column of the text editor

C-style comments delimited by ***/** or **/*** and midline comments delimited by an asterisk will not be color-coded. The T-Spice simulation engine will correctly interpret them as comments, however.

Undo and Redo

The **Edit > Undo** command (**Ctrl+Z** or ↶) reverses changes made to the text of the input file.

Undo reverses the most recent of the editing operations stored in the undo buffer. The previous 100 editing operations are stored in the undo buffer; they are of the following types:

- Typing, including delete and backspace keystrokes.
- Edits made with the **Cut**, **Copy**, **Paste**, or **Clear** commands.
- Edits made with **Insert Command** (see Simulation Commands on page 19).

Undo is unavailable under the following circumstances:

- Immediately after T-Spice is launched.
- Immediately after an input file is created or opened.

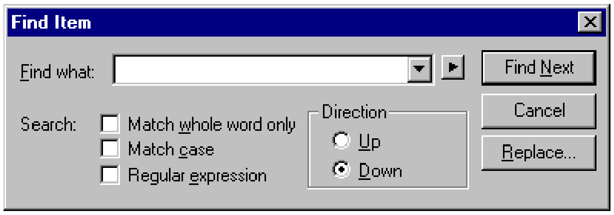
The **Edit > Redo** command (**Ctrl+Y** or ↷) restores changes reversed with a previous **Undo** command. Each of the events stored in the undo buffer can be redone one at a time in reverse order.

Search and Replace

The T-Spice text editor supports string and regular expression search and replace operations.

Find

The **Edit > Find** (**Ctrl+F**) command opens the **Find Item** dialog.



Find what String to be searched for in the netlist file.

Match whole word only Causes T-Spice to find only whole words that match the entered search string.

Match case Causes T-Spice to find only strings whose case matches that of the search string.

Regular expression Activates regular expression searching. See Regular Expression Rules, below for further information.

(Special) Opens a submenu of special character codes that can be inserted into the target string. These codes are prefixed by the caret character (**^**) unless they are UNIX-style regular expressions. Options include:

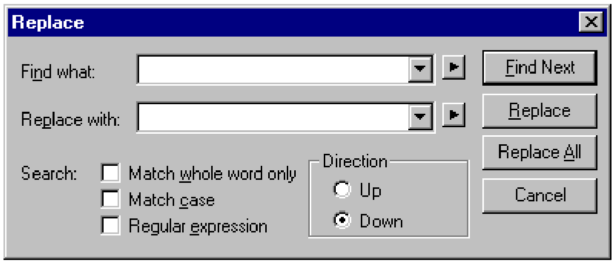
- **Manual Line Break**
- **Tab Break**
- **White Space**
- **Caret Character**

Direction Causes T-Spice to search the netlist file in the desired direction (up or down), relative to the cursor.

Find Next Activates a search for the entered string.

Replace Opens the **Replace** dialog.

The **Replace** dialog provides an extra input field and two additional option:s



Replace with String to replace the search string in the netlist file.

Replace Replaces the next instance of the search string with the replace string.

Replace All Replaces every instance of the search string with the replace string.

Regular Expression Rules

The **Regular expression** option in the **Find Item** or **Replace** dialogs causes T-Spice to interpret the search string as a Unix-style regular expression. Instead of interpreting the caret combinations **^t**, **^l**, and **^w** as special sequences, T-Spice replaces them with Unix-style combinations that use the backslash (\) escape character.

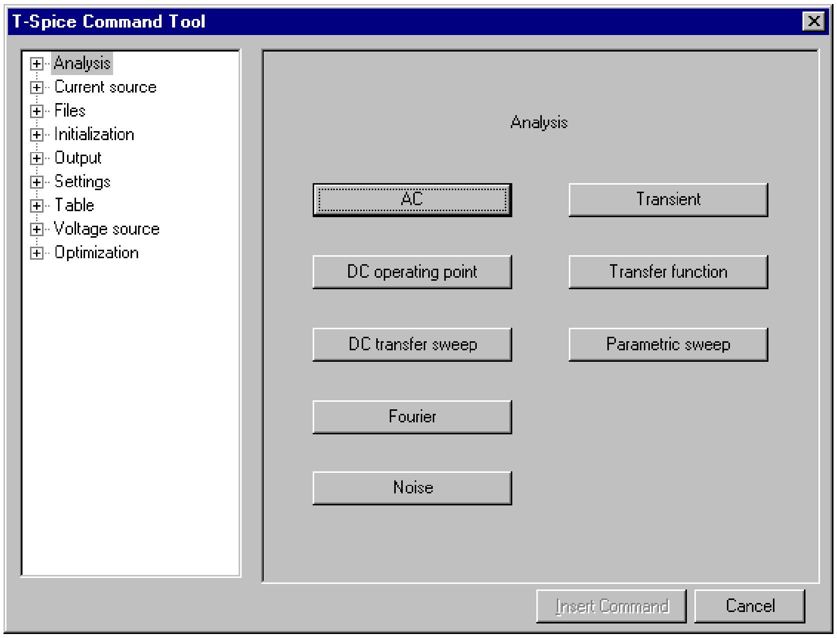
The following table lists the rules T-Spice will follow when searching in regular expression mode:

Syntax	*Description*	*Example*
\n	Line break.	
\t	Tab character.	
^	Beginning of line.	
$	End of line.	
.	Any character except line break.	**p.n** matches **pin** and **pan**.
[ ]	One of the characters enclosed in the brackets.	**p[ai]n** matches **pin** and **pan** but not **pun**.
[^set]	Any character not enclosed in square brackets.	**p[^i]n** matches **pan** but not **pin**.
[*set*]	A set of characters including any character from the set enclosed in square brackets.	**[0-9]** matches any digit. **[spice]** matches any of the characters **(s p i c e)**.
[*set*]*	Zero or more occurrences of the set enclosed in square brackets.	**[0-9]*1** matches **1** and **11** and **381**.
[*set*]+	One or more occurrences of the set enclosed in square brackets.	**[0-9]+** matches **2** and **4532**.
-	Optional match.	**12-** matches **1** and **12**.
\\(	Begin tag.	**A\\([0-9]+\\)** matches **A123** and substitutes **123** for the first tag **(\1)** in the replacement string.
\\)	End tag.	
\*n*	Text matching the *n*th parenthesized component of the regular expression, where *n* is a single digit.	If the search string is **\\(ab\\)\\(cd\\)** and the replacement string is **\2\1**, T-Spice will replace **abcd** with **cdab**.
&	The entire matched regular expression.	If the search string is **Windows** and the replacement string is **MS-&**, T-Spice will replace **Windows** with **MS-Windows**.
\0	The entire matched regular expression.	
\\	Literal backslash	**a\\\\n** matches **a\n**.

Syntax	*Description*	*Example*
\&	Literal ampersand, to avoid expression substitution	If the search string is **123** and the replacement string is **&\&&**, T-Spice will replace **123** with **123&123**.

Command Tool

T-Spice also provides a *Command Tool*, which presents a categorized listing of T-Spice simulation commands and options. You can use this tool as a guide in composing commands for the input file.



The Command Tool lists the T-Spice simulation commands in a hierarchical arrangement, with general categories that expand into individual commands for each category. You expand or collapse a category by double-clicking it or clicking the plus or minus sign next to the category. When you single-click a category name in the left-hand tree, buttons representing individual commands appear in the right-hand pane of the dialog.

You select a command by clicking its button or its name in the tree; when you do, T-Spice presents the appropriate options in a set of fields in the right-hand panel of the dialog.

For a full description of this tool, see Simulation Commands on page 19.

Simulating a Design

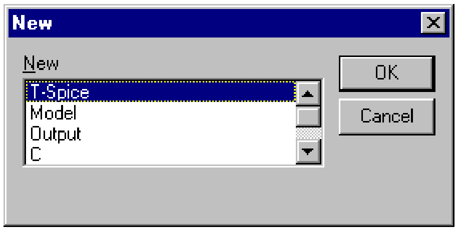
This section illustrates some basic principles of T-Spice operation using a simple example.

Creating a New Input File

Create a new input file by clicking the **New File** button (🗋) or selecting **File > New**.

When you click the **New File** button, T-Spice opens a new text window with the default filename **T-Spice1**. T-Spice increments the default name for additional new files, assigning them **T-Spice2**, **T-Spice3**, etc.

When you select **File > New**, a dialog appears that prompts you to specify the type of file you wish to create:



The extension shown in parentheses will be the default file format.

- T-Spice (**.sp**)
- Model (**.md**)
- Output (**.out**)
- C (**.c**)
- Header (**.h**)
- Text (**.txt**)

Selecting **T-Spice** enables color-coding for T-Spice commands and comments. An empty active window appears in the display area; it is provisionally called **T-Spice1** (or **T-Spice2**, **T-Spice3**, etc.). Change the name to **test.sp** as follows:

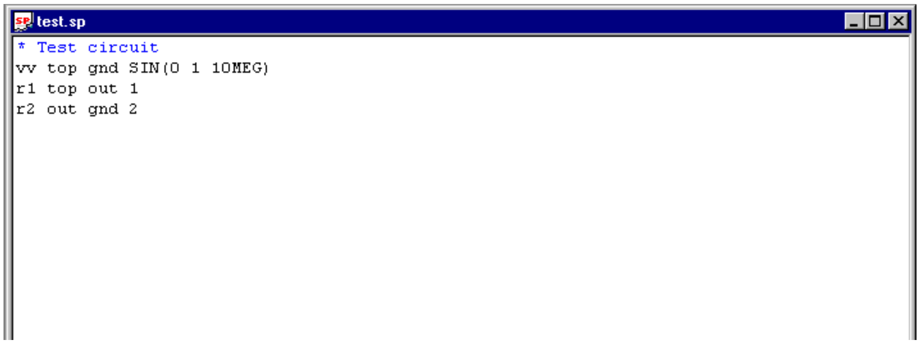
- Use the **File > Save** command (or click 💾 in the toolbar).
- Type **test.sp** under **File name** in the **Save As** dialog.
- Press **Return** or click **OK**.

T-Spice will change the window title accordingly.

Entering the Circuit Description

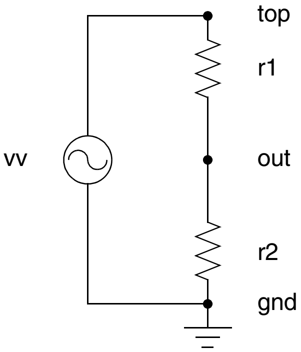
Device Statements

Type the following text in the **test.sp** window, pressing **Enter** after each line:



```
* Test circuit
vv top gnd SIN(0 1 10MEG)
r1 top out 1
r2 out gnd 2
```

The first line is a *comment*. (T-Spice always treats the first line of an input file as a comment, whether or not it begins with a comment symbol.) The next three lines, beginning with **vv**, **r1**, and **r2**, are *device statements* comprising a SPICE description of the elementary voltage divider schematically represented below.

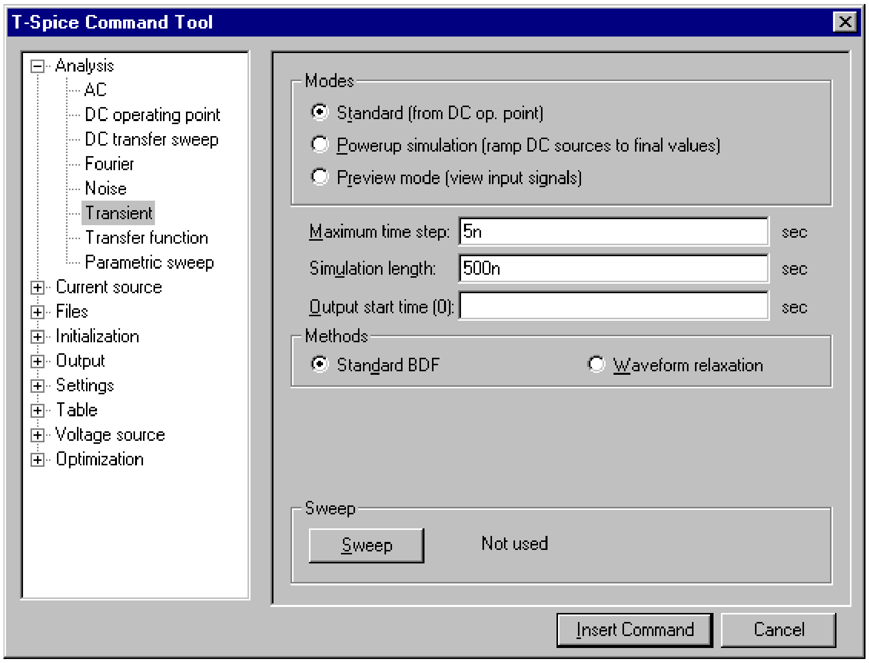


For information on comments, see Input Conventions on page 42. For information on device statements, see Device Statements on page 141.

Simulation Commands

With the blinking cursor at the beginning of the next blank line in the **test.sp** window, use the **Edit > Insert Command** command (or click in the toolbar) to open the *Command Tool*.

The first category on the left is **Analysis**. Expand that category and select the **Transient** command. T-Spice displays the appropriate options in the right-hand pane of the Command Tool.



Modes

● Standard (from DC op. point)

○ Powerup simulation (ramp DC sources to final values)

○ Preview mode (view input signals)

Maximum time step: 5n sec

Simulation length: 500n sec

Output start time (0): sec

Methods

● Standard BDF ○ Waveform relaxation

Sweep

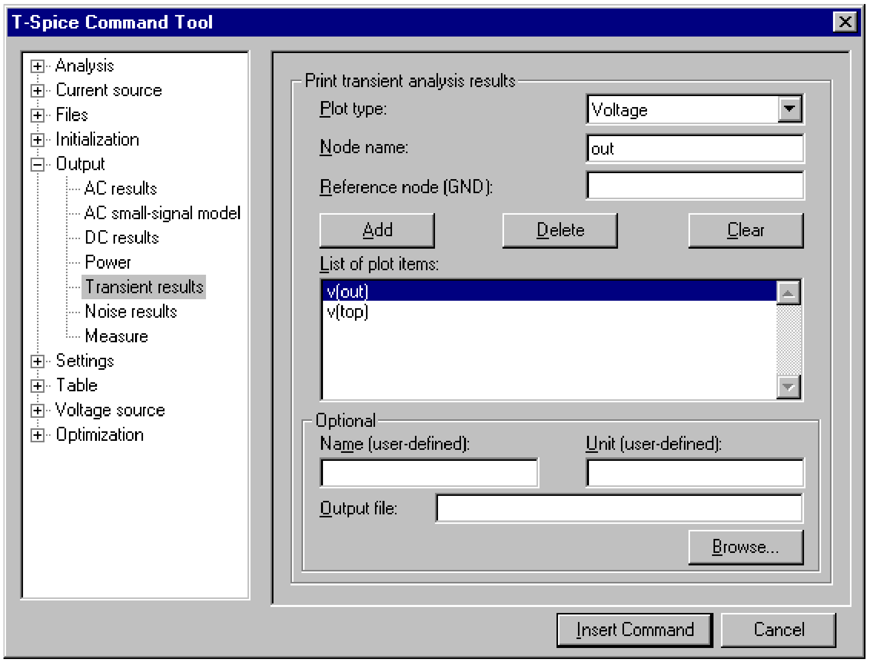
Not used

In the Command Tool:

- For **Maximum Time Step** enter **5n**.
- For **Simulation Length** enter **500n** (nanoseconds).
- Click **Insert Command**.

The full command is placed as a line of highlighted text into **test.sp**, and the Command Tool closes.

Click at the end of this line to deselect it, then press **Enter** to create another blank line. Open the Command Tool again, and select the **Output** category and the **Transient results** command.

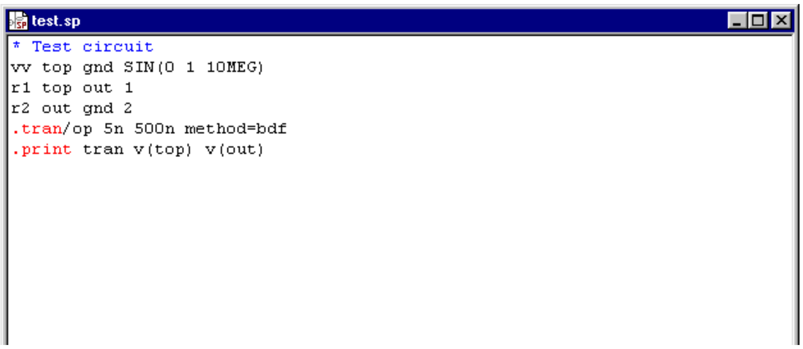


Print transient analysis results
Plot type: Voltage
Node name: out
Reference node (GND):
List of plot items:
v(out)
v(top)
Name (user-defined):
Unit (user-defined):
Output file:

In the Command Tool:

- For **Plot type**, select **Voltage**.
- For **Node name** enter **top**. Then click the **Add** button.
- Replace the name **top** with the name **out**, and click the **Add** button again.
- Click the **Insert Command** button.

test.sp now looks like this:



```
* Test circuit
vv top gnd SIN(0 1 10MEG)
r1 top out 1
r2 out gnd 2
.tran/op 5n 500n method=bdf
.print tran v(top) v(out)
```

The last two lines (beginning with **.tran** and **.print**) are the simulation command sequence, directing T-Spice to perform a transient analysis for 500 nanoseconds with a maximum time step of 5 nanoseconds, and to report the results of the transient analysis for the voltages at nodes **top** and **out**.

Alternatively, simulation commands can be incorporated directly into a circuit schematic. Then the commands will be present in the netlist when it is exported.

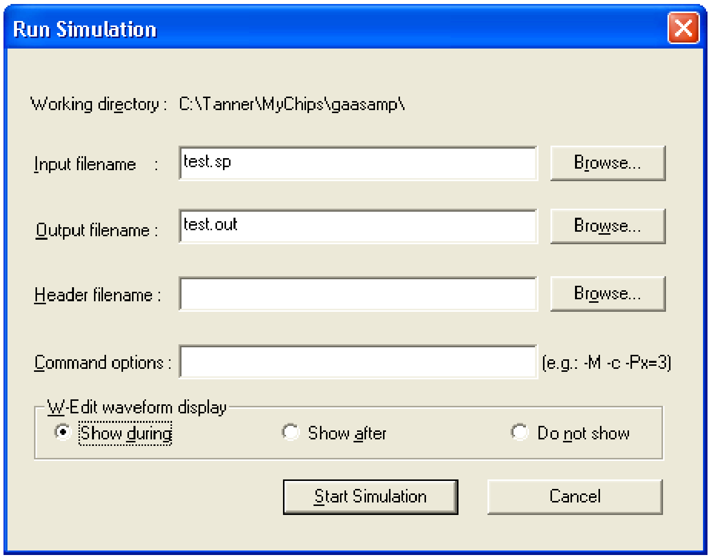
For information on the commands used in this example, see Simulation Commands on page 50.

Save **test.sp**. The circuit description is now complete and the simulation can be run.

Running the Simulation

Use the **Simulation > Run Simulation** command (or click ▶ in the toolbar) to initiate simulation.

A dialog appears, suggesting an output file name: **test.out**. (You can change this name if you prefer.) The waveform display option **Show during** allows dynamic visualization of the results in W-Edit.


Run Simulation

Working directory : C:\Tanner\MyChips\gaasamp\

Input filename : test.sp

Output filename : test.out

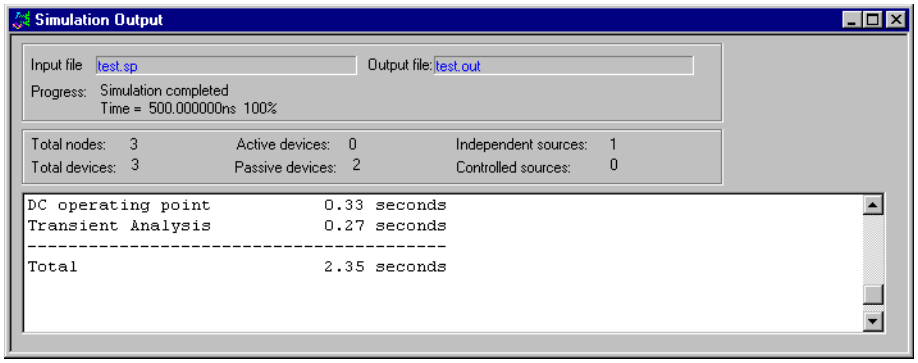
Header filename :

Command options : (e.g.: -M -c -Px=3)

W-Edit waveform display: ● Show during ○ Show after ○ Do not show

Click **Start Simulation** (or press **Enter**).

The status of the simulation is shown in the Simulation Status window.


Simulation Output

Input file test.sp Output file: test.out

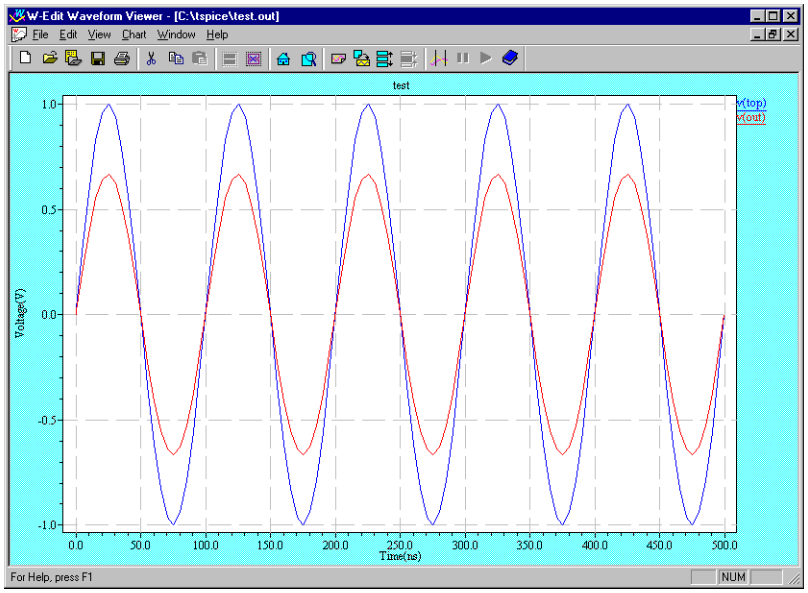
Progress: Simulation completed
Time = 500.000000ns 100%

Total nodes: 3 Active devices: 0 Independent sources: 1
Total devices: 3 Passive devices: 2 Controlled sources: 0

```
DC operating point          0.33 seconds
Transient Analysis          0.27 seconds
-----------------------------------------
Total                       2.35 seconds
```

Text within the scrolling portion of the Simulation Status window can be copied for pasting into another input window.

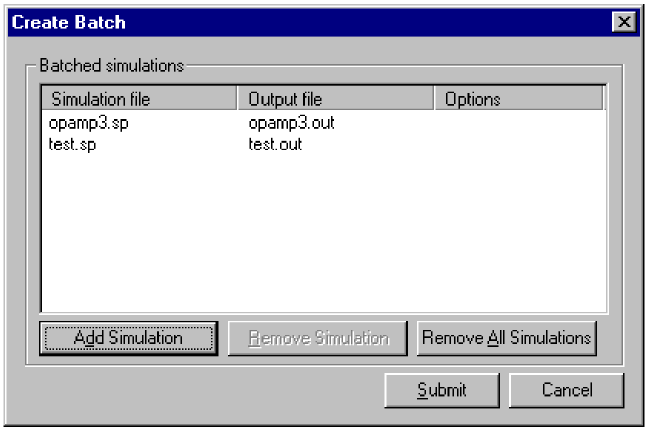
At the same time, the output of the simulation is shown graphically in a separate W-Edit window. For information on W-Edit, see the *W-Edit User Guide*.



The voltage at node **out** is, correctly, two-thirds of the voltage at node **top**.

Simulation Queueing

You can queue simulations for automatic serial execution by choosing **Simulation > Batch Simulations** or clicking the **Batch Simulations** button ( ) in the Simulation toolbar.



Simulation file	Output file	Options
opamp3.sp	opamp3.out	
test.sp	test.out	

Add Simulation Opens the **Add Simulation** (page 512) dialog to add a new file to the list of files to be submitted to the Simulation Manager queue.

Remove Simulation Removes the selected file from the list of files to be submitted to the Simulation Manager queue.

Remove All Simulations Removes all files from the list of files to be submitted to the Simulation Manager queue.

Submit Sends the files listed in the **Create Batch** dialog to the **Simulation Manager** queue.

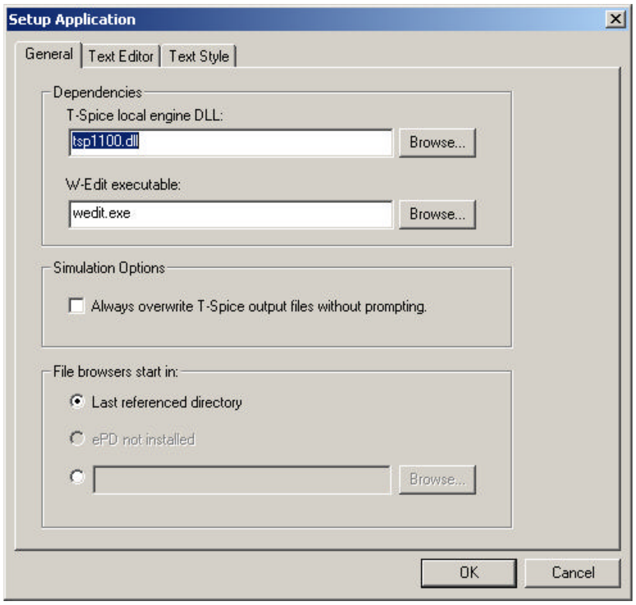
You can also run batch simulations from a DOS or Unix command-line using T-Spice’s -batch option, which passes the simulation commands listed in a text batch file to the T-Spice engine. See Command-Line Options on page 28 for a description of command-line options.

Setup Options

The three tabs in **Setup > Application** control global options in T-Spice.

General

Warning: Changing dependency values is not recommended.



T-Spice local engine DLL: tsp1100.dll

W-Edit executable: wedit.exe

□ Always overwrite T-Spice output files without prompting.

● Last referenced directory

○ ePD not installed

○

Dependencies

Unless you need a specific patch or model we recommend using the default dependencies.

- **T-Spice local engine DLL:** Enter or browse to a DLL file name if you would like to use a different version of the model definitions.
- **W-Edit executable**: allows you to select a different version of the waveform viewer to use in conjunction with T-Spice.

Simulation Options

When this box is checked, T-Spice will **Always overwrite T-Spice output files without prompting**.

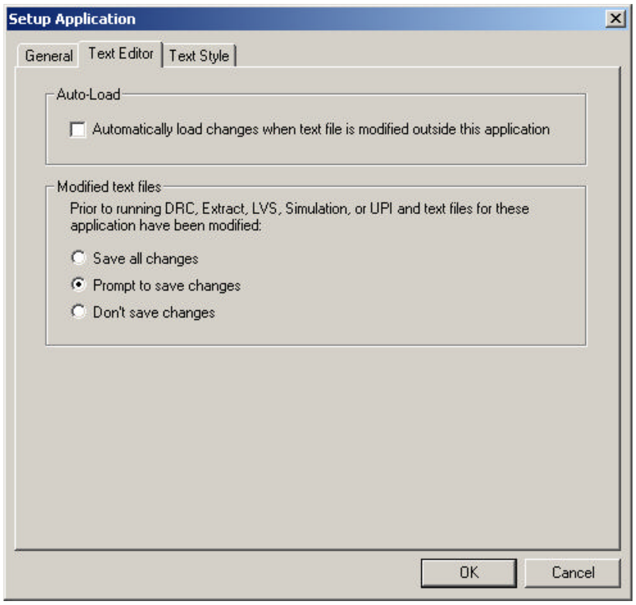
File browsers start in:

Sets the default directory for file open and file save operations.

- **Last referenced directory**—Sets the initial path location of all file browsers to the last directory in which a file was opened or saved. (This is the default option.)
- **ePD active project**—Sets the initial path location of all file browsers to the directory of the project active in the ePD Dashboard so that file operations can be synchronized with the ePD/ViewDraw active project directory. (This option will be disabled if ePD is not installed.)
- (unlabeled)—Sets a fixed initial path which you can type in or browse to.

Text Editor

Determines T-Spice behavior in file saving operations. The most recently used settings are kept as defaults.



Auto-Load

☐ Automatically load changes when text file is modified outside this application

Modified text files

Prior to running DRC, Extract, LVS, Simulation, or UPI and text files for these application have been modified:

○ Save all changes

● Prompt to save changes

○ Don't save changes

Options include:

Auto-Load

When checked, T-Spice automatically loads changes whenever a text file is modified outside T-Spice. No warning will be provided.

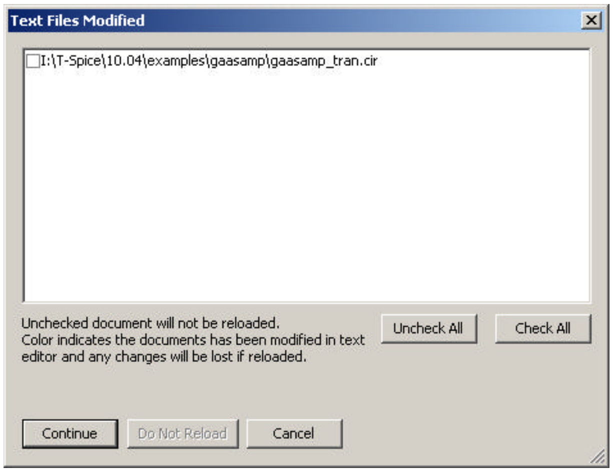
Modified text files

Controls how files modified within T-Spice will be saved prior to simulation operations. Options are:

- **Save all changes**—automatically saves all active windows when one of the above operations is invoked.
- **Prompt to save all changes**—T-Spice will display a prompt when there are unsaved changes in the simulation input file. Select **Yes** to save the input file and proceed with simulation. If you select **No**, the **Run Simulation** command is ignored.
- **Don't save changes**—modified files will not be saved and the operation will use the stored version of those files.

When Auto-Load is disabled, T-Spice will open a checklist of all the files open in the text editor that have been modified elsewhere. You will have the option to reload modified files (checked) or not.

Files that have also been modified in the text editor will be highlighted. Similarly, when **Prompt to save changes** is selected, T-Spice will open a checklist of the modified files associated with the operation you are running. For example, you might open a text file in T-Spice, then later overwrite it by reexporting a netlist. In such a case, T-Spice will display the following message to warn you that the file on disk has changed since you opened it in T-Spice:



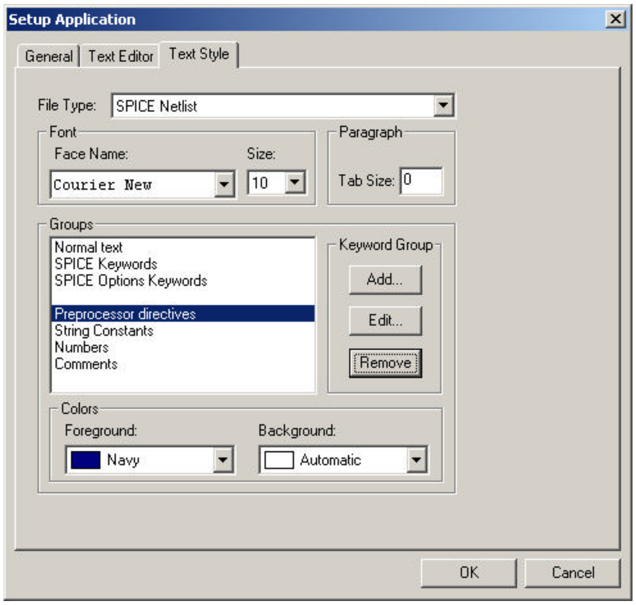
□I:\T-Spice\10.04\examples\gaasamp\gaasamp_tran.cir

Unchecked document will not be reloaded.
Color indicates the documents has been modified in text editor and any changes will be lost if reloaded.

If you click **Yes**, T-Spice updates the file in memory.

Text Style

Use this tab to define the types of text (**keyword** groups) that will be highlighted when displayed in the T-Spice text editor, and their formatting. You can set text formats for SPICE netlists, C macro and text files.



File Type: SPICE Netlist

Face Name: Courier New

Size: 10

Tab Size: 0

Normal text
SPICE Keywords
SPICE Options Keywords

Preprocessor directives
String Constants
Numbers
Comments

Foreground: Navy

Background: Automatic

Each file type has a set of predefined keyword groups that cannot be edited or deleted. Use this tab to view those settings, and to add or remove your own keyword groups with customized characteristics.

File Type

A drop down list of the file types for which keywords are or can be defined.

Font

Allows you to set the typeface (**Face Name**) and point **Size** in which a given keyword group will appear.

Paragraph

Allows you to set the increment, in spaces, of the **Tab Size** used by the text editor.

Groups

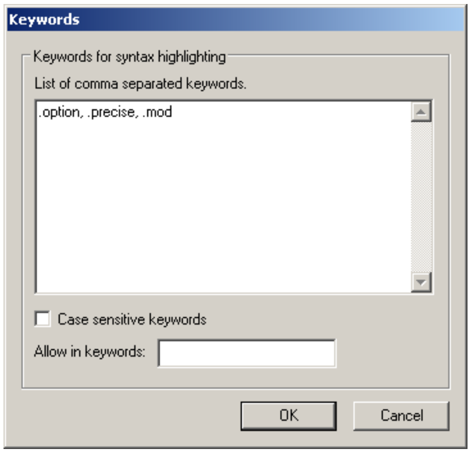
Displays the keyword groups defined for a given file type. Use **Add** to enter the name of a new keyword group. Use **Edit** to enter the keywords belonging to a group. Use **Remove** to delete a keyword group.

Colors

Use **Foreground** and **Background** to set the respective colors for a keyword group.

Adding Keywords to a Group

Edit keyword groups opens the **Keywords** dialog, which allows you to enter keywords and to specify whether the case is evaluated (**Case sensitive keyords** checkbox enabled) when highlighting is applied.



List of comma separated keywords.

.option, .precise, .mod

□ Case sensitive keywords

Allow in keywords:

Command-Line Options

T-Spice supports command-line options that allow you to alter simulation commands or options without changing the input netlist. There are two ways to use command-line options in T-Spice:

- You can use any command-line option in conjunction with the **tspcmd.exe** executable to run simulations in a command-line environment, such as DOS or Unix.
- Most command-line options are also available within the T-Spice user interface; you can type them in the **Simulation > Run Simulation** dialog before you begin a simulation.

Command descriptions follow these conventions:

- Variables to be replaced by actual names, numbers, or expressions are indicated by ***italics***.
- Square brackets **[ ]** enclose items that are *not required*. The brackets should *not* be typed on the command line.

Some command-line options allow you to specify an included or referenced file. Use the following rules to specify filenames in the command line:

- Specify filenames relative to the working directory, or using a fully qualified pathname. In the T-Spice **Run Simulation** dialog, the **Working Directory** field identifies your main input file directory. (See Simulation > Run Simulation on page 511.)
- Filenames containing spaces must be enclosed in single or double quotes.

T-Spice supports the following command-line options:

Option	*When on*	*When off (default)*
-batch *batchfile* (Not available in T-Spice GUI.)	Processes the simulations listed in the text file ***batchfile*** in series. See Running T-Spice From the Command-Line on page 30 for ***batchfile*** syntax. This option is only available with the **tspcmd.exe** executable.	—
-c	Estimate MOSFET drain and source areas and perimeters from MOSFET lengths and widths. Equivalent to **.options** (page 95) **moscap=1**.	Set drain and source areas (if not specified in MOSFET statements) to zero.
-C	Disable the connectivity check.	Enable the connectivity check.

-d [*probefile*]	Turns on probing and optionally specifies the binary output filename. This option is equivalent to adding the following commands to the input netlist:
	`.probe`
	`.probe noise dn(*,tot)`
	`[.options probefilename=filename]`
	If a filename is specified, it will override the command **.options probefilename** in the netlist.
		No probing commands are specified, unless explicitly stated in the input netlist.

-dll ***dllfile*** (Not available in T-Spice GUI.)	Runs the current simulation using the specified DLL dllfile. See Running T-Spice From the Command-Line on page 30 for a description. This option is only available with the **tspcmd.exe** executable.	Use the default DLL (recommended).
-h *headerfile*	Specify a header file to be processed at the start of T-Spice simulation.	Parse only the contents of the current T-Spice input file.
-i “*TSCommand*”	Executes the T-Spice command enclosed in double quotes. Input commands added with **-i** are parsed after the header file, but before other input files.	—
-l	Echoes all parsed input lines to the Simulation Status window or stderr output. This is equivalent to setting **.options echo=1**.	Input lines are not shown in simulation output.

Option	*When on*	*When off (default)*
-m *modelfile*	Include model definitions from ***mfile***. Equivalent to the **.model *mfile*** command.	Use model parameters specified in the input file, if available.
-n	Disables splash screen.	Splash screen appears on application startup.
-o *outfile*	Print results to ***ofile***.	Print results to standard output (the screen or the Simulation Window).
-P *parameter* =*value*	Assign a ***value*** (a plain number or an expression enclosed by single quotes) to ***parameter***. Parameter assignments made with this option override assignments made with the **.param** (page 102) command in the input file. This option can be used as many times as desired.	Use parameters specified in the input file, if any.
-q	"Quiet" mode: disable the simulation status display.	Enable the simulation status display.
-U	Print a usage message and quit.	—
-V	Print the version number and acknowledgments and quit.	—
netlistfile	Use ***netlistfile*** as the circuit description. Netlists generally have the extension **.sp** or **.cir**.	[The argument is required when running **tspcmd.exe**.]

Running T-Spice From the Command-Line

The **tspcmd.exe** executable file allows you to run the T-Spice engine, without a graphical interface, from a command-line environment such as DOS or Unix. To run T-Spice from a command-line, use the following syntax:

```
tspcmd [-aqlCUV] [-dll dllfile] [-batch batchfile]
       [-d [probefile]] [-P parameter=value]
       [-m modelfile] [-o outfile] [-i "command"] netlistfile
```

The options **-batch** and **-dll**, are available *only* in command-line environments; you cannot use these options in the T-Spice **Run Simulation** dialog.

Changing the Engine DLL from the Command Line

The command-line option:

```
-dll dllfile
```

changes the T-Spice engine local DLL to the specified file for the current simulation. The default DLL is specified in the T-Spice user interface dialog, **Options > Dependencies**. Changing DLL files is not recommended.

Batch Simulations from the Command Line

The command-line option:

```
-batch batchfile
```

allows you to pass a list of simulations to T-Spice for serial execution. The ***batchfile*** specifies a text file that contains a list of simulations and accompanying command-line options. Each simulation and its associated options must be listed on a separate line, using the following syntax:

```
[-acqruCMTUV] [-dll dllfile] [-d probefile]
[-P param=value][-m modelfile][-o outfile]
[-i “TScommand”] netlistfile
```

3 Simulation Concepts

Simulation Algorithms

T-Spice is designed to solve a wide variety of circuit problems. Its flexibility is due to robust *algorithms* which can be optimized by means of user-adjustable *parameters*. This chapter contains an overview of T-Spice's algorithms and parameters.

In what follows, when reference is made to an "*option*" (such as the **numnd** option) it means that the corresponding quantity is controlled with the **.options** command. For information on the **.options** command, see Simulation Commands on page 50.

Kirchoff's Current Law

T-Spice uses Kirchoff's Current Law (KCL) to solve circuit problems. To T-Spice, a *circuit* is a set of *devices* attached to *nodes*. The circuit's state is represented by the voltages at all the nodes. T-Spice solves for a set of node voltages that satisfies KCL (implying that the sum of the currents flowing into each node is zero).

In order to evaluate whether a set of node voltages is a solution, T-Spice computes and sums all the currents flowing out of each device into the nodes connected to it (its *terminals*). The relationship between the voltages at a device's terminals and the currents through the terminals is determined by the *device model*. For example, the device model for a resistor of resistance R is $i = \Delta v / R$, where Δv represents the voltage difference across the device.

DC Analysis

Most T-Spice simulations start with a DC operating point calculation. A circuit's *DC operating point* is its steady state, which would in principle be reached after an infinite amount of time if all inputs were held constant. In DC analysis, capacitors are treated as open circuits and inductors as short circuits.

Because many devices, such as transistors, are described by nonlinear device models, the KCL equations that T-Spice solves in DC analysis are nonlinear and must therefore be solved by iteration. On each iteration, T-Spice tries to find a set of node voltages that satisfies KCL more closely than the previous set. When the KCL equations are satisfied "well enough" (the sums of currents into nodes are small enough), the process stops.

The **abstol** and **reltol** options determine how closely KCL must be satisfied. The **numnd** option imposes a limit on the number of iterations. If **numnd** iterations are reached without a solution being found, then nonconvergence is declared.

T-Spice sometimes uses a technique called *source stepping* to find a circuit's DC operating point. In source stepping, all voltage and current sources are ramped up from zero to their final values. This allows T-Spice to find the DC operating points of difficult-to-converge circuits. Source stepping is used only in non-converging cases of initial DC operating point calculations, and not during DC analysis sweeps. The smallest source step that T-Spice will take is controlled by the **minsrcstep** option.

g_{min} Stepping

Some non-convergence errors can be eliminated by ensuring a sufficient conductance across capacitors. The option **gmindc** specifies a conductance that is added in parallel with all *pn* junctions during DC analysis. T-Spice applies the **gmindc** conductance to various elements as follows:

- diode—conductance is added across the positive/negative terminals.
- BJT—conductance is added across the base/emitter and the base/collector terminals.
- MOSFET—conductance is added across the source/bulk, drain/bulk, and the source/drain terminals.
- MESFET—conductance is added across the source/gate, drain/gate, and source/drain terminals.

The default value for **gmindc** is 10^{-12}.

When a DC operating point non-convergence occurs, T-Spice can begin a g_{min} stepping algorithm to find the minimum conductance that yields a convergent solution. . The g_{min} stepping algorithm is triggered when a non-convergence occurs and the value of option **gramp** is greater than zero. Together, the options **gmindc** and **gramp** specify a search range for the minimum required conductance, g_{min}:

$\textbf{gmindc} \le g_{min} \le \textbf{gmindc} \cdot 10^{\textbf{gramp}}$

T-Spice's g_{min} stepping algorithm searches the specified conductance range in two steps. First, T-Spice performs a binary search between **gmindc** and $\textbf{gmindc} \cdot 10^{\textbf{gramp}}$. T-Spice searches for the smallest value of g_{min} that results in a converged solution. T-Spice automatically ends the binary search when it reaches a Δg_{min} that is less than or equal to a factor of 10.

Starting with binary search results, T-Spice then begins reducing the value of g_{min} by a factor of 10 in each iteration. Once a non-convergence occurs, the previous convergent iteration provides the final solution.

Transient Analysis

In transient analysis, T-Spice solves for a circuit's behavior over some time interval. In this mode, T-Spice takes small time steps, solving for the circuit's state at each step. At each time step, two approximations are made.

First, a small error — the *discretization* error — is introduced because T-Spice cannot take infinitely small time steps. The **chargetol** and **relchargetol** options determine the acceptable limits of discretization error. In general, taking smaller time steps decreases the discretization error, so tightening the tolerances has the effect of higher accuracy at the expense of smaller time steps and therefore longer simulation times and larger output files. The discretization error is also affected by the order of the time integration method used, adjusted with the **maxord** option.

Second, just as in DC analysis, T-Spice solves the nonlinear KCL equations iteratively at each time step. The accuracy is affected by an iteration stopping criterion. The same tolerances as in DC analysis — **abstol** and **reltol** —affect this solution process. The iteration count limit for a transient analysis time step is **numnt**, which is typically much smaller than **numnd**; the previous time step always provides a good initial guess for a transient analysis Newton iteration, so that fewer iterations are typically required than for DC analysis, where a good initial guess is usually not available. Another iteration limit, **numntreduce**, affects time step selection after a successful time step. If T-Spice took less than **numntreduce** iterations to find the solution at a time step, the next time step is adjusted (often increased) according to the discretization error tolerances **chargetol** and **relchargetol**. But if the number of iterations required is between **numntreduce** and **numnt**, then the time step is always decreased on the next step (even if the step was successful).

As in DC analysis, some non-convergence errors can be avoided by adding a small conductance across capacitors. For transient analysis, the option **gmin** specifies a conductance that is added in parallel with all *pn* junctions. The default value of **gmin** is 10^{-12}.

Trapezoidal Integration Method

T-Spice's default method for transient analysis uses trapezoidal integration with the lvltim=1 delta-voltage time step control algorithm.

The trapezoidal formula calculates the average slope of the present and next time point to approximate the value of the integral of the differential equations used in the time range calculations, as follows in simplified form. The following approximation is used to discretize the differential equation:

$$V_{n+1} = V_n + \frac{h}{2}\left(\frac{dV_{n+1}}{dt} + \frac{dV_n}{dt}\right) \tag{0.1}$$

where

$$\begin{aligned} V_{n+1} &= \textit{present unknown voltage value} \\ V_n &= \textit{previous time-point solution} \\ h &= \textit{time step length} \\ n &= \textit{time interval} \end{aligned} \tag{0.2}$$

Gear's BDF Method

T-Spice's alternate method for transient analysis uses Gear's backward differentiation formulas (BDF). In this method, the time derivative of charge in the KCL equations is replaced by an approximation involving the solution at the last few time points. The first-order BDF method uses only one previous time point, and it is equivalent to the well-known Backward Euler method. In this method, the discretization error is a linear function of the step size. The second order method uses two previous time points, and its discretization error is proportional (for small time step sizes) to the time step size squared. In general, the *k*th order BDF method uses *k* previous time points.

T-Spice uses a variable-step-size, variable-order, and variable-coefficient implementation of the BDF method. T-Spice automatically adjusts the time step size and BDF order (between 1 and 4) to minimize the number of time steps required to meet the given error tolerances. The maximum order used can be adjusted with the **maxord** option. The variable-coefficient implementation was chosen over the fixed-coefficient and fixed-leading-coefficient methods because it offers the best stability properties, especially with frequently varying time step sizes.

At each time step, the BDF discretization results in a nonlinear system of equations (representing KCL) which is solved iteratively as described above. If the iteration succeeds, the discretization error is examined (by comparison with an explicit predictor). For example, in the order 1 case, the difference between the Forward Euler predictor and the computed BDF (Backward Euler) solution provides a bound on the discretization error. If the error is within the prescribed tolerance (defined by **chargetol** and **relchargetol**), the step is accepted, and the error is used to adjust the step size for the next time step. If the error is too large, the time step is rejected and reattempted with a smaller step size. This will produce answers which approach a more stable numerical solution. Gear integration often produces superior results for power circuitry simulations, due to the fact that high frequency ringing and long simulation periods are often encountered.

Small-Signal Analysis

Some of T-Spice's analysis commands use *small-signal* models. Small-signal analysis linearizes the KCL equations about an operating point. Subsequent computations are performed on the linearized circuit, which can be solved in one matrix-vector operation. In AC analysis, for example, one matrix-vector solve is done at each frequency point to find the AC solution; no iteration is necessary. The linearized small-signal model is valid *only locally*, so if the operating point changes, then a new linearized model has to be computed.

Tolerances

T-Spice's simulation speed and accuracy are controlled by various tolerance values. Computer-based simulators like T-Spice solve circuit equations using finite precision arithmetic. This means that numerical approximations are made at several steps of the solution process. The errors introduced by these approximations in T-Spice are bounded by tolerance settings.

Each approximation is controlled by a relative tolerance *trel* and an absolute tolerance *tabs*. In most cases, the relative tolerance is used — the approximation error may not exceed *trel* × |*v*|, where *v* is the value of the quantity to be approximated. The absolute tolerance is used when the approximated quantity's value is close to zero; in that case, the error may not exceed *tabs*. In general, the error must be less than the maximum of *trel* × |*v*| and *tabs*.

abstol — reltol

Corresponding to each node in a circuit is an equation which expresses Kirchoff's Current Law (KCL), according to which the branch currents flowing into the node must be zero. The actual sum of these branch currents at a node is called the "residual current" for that node, and it is a function of all node voltages. In a sense, then, the value of this current at a given node is a measure of how well KCL holds at that node. T-Spice attempts to find a solution (a set of node voltages) that causes KCL to be satisfied at all nodes. The correctness of the solution is measured by the norm of the vector of residual currents at all nodes.

In general, the KCL equations are complicated and nonlinear, and T-Spice solves them by numerical iteration. Each iteration results in an improved approximation of the true solution of the KCL equation, in the sense that the norm of the residual currents decreases with each iteration. However, it is often impossible to make the residual current exactly zero, because of the finite precision arithmetic used. Even if infinite precision arithmetic were available, it might take infinitely many iterations to make the residual current zero. Therefore, the iteration is considered to have converged to a solution when the residual current is within the tolerances defined by **abstol** and **reltol**. Thus, **abstol** and **reltol** control how small the residual currents must be — how far from satisfying KCL the nodes can be — before the system is considered solved.

To be more precise, **reltol** and **abstol** are applied as relative and absolute tolerances, as described above, to the residual currents at each node. The tolerance used at a particular node is max(**abstol**, **reltol**×*imax*), where *imax* is the largest branch current (in absolute value) flowing into the node in question. The **abstol** and **reltol** tolerances are used whenever T-Spice solves the KCL equations for a DC solution or a transient analysis time step.

The default value for **reltol** is 1 × 10^{-4}, or 0.01%. The default value of **abstol** is 0.5 nanoamps. These values should be reduced for sensitive analog designs.

numnd — numnt — numntreduce

numnd defines the maximum number of iterations allowed during the solution of the KCL equations during a DC analysis. If, after **numnd** iterations, the **abstol/reltol** tolerances have not been satisfied, the iteration is considered to have failed and nonconvergence is declared. T-Spice then stops the iteration and reacts appropriately:

- On initial DC operating point calculations, source stepping is invoked.
- During DC transfer analysis, the transfer step size is reduced.

During source stepping, the maximum number of iterations for each source step is **numnd** ⁄10. If **numnd** ⁄10 iterations are exceeded during source stepping, the source step size is reduced. If the minimum source step size (**minsrcstep** option) is violated, then nonconvergence is declared for the DC operating point computation.

If the initial DC operating point computation fails at the beginning of a transient analysis, T-Spice attempts a powerup simulation. In a powerup simulation, all voltage and current sources are slowly ramped up from zero to their actual values. The default ramp period is 0.1% of the transient simulation final time, and it can be overridden using the **poweruplen** option.

numnt determines the maximum number of iterations allowed during the solution of the KCL equations for a transient analysis time step. Another option, **numntreduce,** is used in the time step size after a successful time step. Together, **numnt** and **numntreduce** work as follows. If a time step requires more than **numnt** iterations, the iteration is considered to have failed, and the same time step is reattempted with a smaller step size. If fewer than **numntreduce** iterations are needed for a time step, the next time step is adjusted (often increased) according to the discretization error tolerances (**chargetol** and **relchargetol**). If the number of iterations required is between **numntreduce** and **numnt**, the step size for the next time step is always reduced.

The number of iterations it takes to converge to a circuit solution depends heavily on the circuit to be simulated. Stable circuits with well-defined, steady-state conditions generally require fewer iterations, whereas circuits with poorly defined or unstable steady-state conditions require more iterations. Convergence is also sensitive to the starting point (initial guess) of the iteration. If the starting point is close enough to the operating point, then the iteration will eventually converge (although it might take many iterations in some cases). But if the starting point is not close to the solution, then the iteration may not converge at all. Different starting points (initial guesses) may be specified with the **.nodeset** command. For information on the **.nodeset** command, see Simulation Commands on page 50.

chargetol — relchargetol

Kirchoff’s Law contains a term that represents the time derivative (rate of change) of charge. In transient simulations, T-Spice must replace this derivative by a divided difference approximation. The approximation becomes more accurate as the time step size decreases. T-Spice chooses its time step size so that the error caused by this approximation remains below another tolerance called the “charge tolerance.” You can specify both an *absolute* and a *relative* charge tolerance with the **chargetol** and **relchargetol** options. By decreasing (or increasing) these tolerances, you can force T-Spice to take smaller (or larger) time steps if you believe that the results of transient simulation are not accurate enough (or too accurate for your time constraints).

The value of **relchargetol** is by default the same as **reltol**, so that **reltol** controls the overall relative simulation accuracy. For example, if you would like 0.001% accuracy, simply set **reltol** to 1×10^{-5}; this also sets **relchargetol** to 1×10^{-5}. The **relchargetol** option should be used only to override the general relative tolerance **reltol**.

In general, the KCL tolerances **abstol** and **reltol**, as well as the charge tolerances **chargetol** and **relchargetol**, trade off speed for accuracy such that tightening these tolerances increases simulation accuracy at the expense of speed. However, if the KCL tolerances are too loose relative to the charge tolerances, the simulator may take small time steps because of numerical noise introduced by residual currents. These residual currents exist only because the KCL equations are not being solved exactly, but they may cause the charge tolerances to be violated, leading to excessively small time steps. If this happens, reducing **abstol** and/or **reltol** will result in larger time steps and faster (as well as more accurate) simulation.

Device Model Evaluation

T-Spice can evaluate device models with any of the folowing methods. Direct model evaluation is the default.

Evaluation Methods

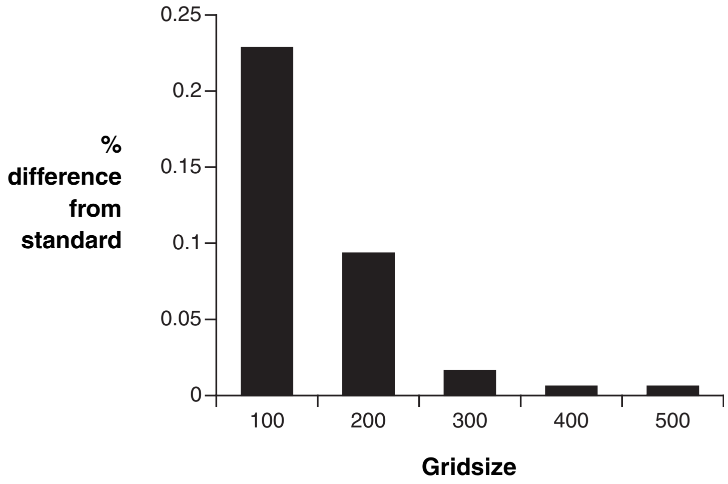
- In *direct* model evaluation, data points (the charges and currents at device terminals) are computed “directly” at each step from terminal voltages using analytical model equations. This method typically produces the most accurate results, but also typically takes the most time, because values must be repeatedly recomputed from very complex equations.
- In *table-based* model evaluation, data points are read or interpolated from precomputed tables stored in memory. When the simulation requires the charge or current value (output) corresponding to a given voltage (input) for a particular device, T-Spice uses the voltage to look up the appropriate value in the device table, interpolating if necessary to arrive at the needed charge or current. This method is considerably faster than direct model evaluation. By default, T-Spice uses analytical models to generate cached, or internal, tables for use during a simulation. At the conclusion of the simulation these internal tables are discarded.
- Another mode of table-based evaluation is *external tables* (see External Tables on page 476). In this mode, “external” tables are generated and stored *prior* to running a simulation. External tables can be used in multiple simulations without having to be regenerated each time; they can be constructed from existing analytical models or associated with *macromodel* elements to simulate

The tables are *internal* because T-Spice creates and maintains them during the course of a simulation and discards them at the end, without user intervention. When a simulation begins, T-Spice starts to construct charge and current tables point by point. By default, tables are computed at resolutions predefined for various device types, but these default resolutions may be overridden by **.gridsize** commands in the input file. T-Spice requires separate tables for devices of different types, devices of the same type with different physical characteristics, or devices of the same type using different models. As T-Spice requires device charges and currents during simulation, it looks them up (or interpolates them from nearby data) in the charge and current tables.

Table Resolution

Table resolution is measured by the number and spacing of the table’s *grid points*. Grid point spacing is adjusted dynamically, with new values computed analytically when necessary, and other values interpolated between existing ones. However, it is possible to force an inappropriate number of grid points or equal spacing between grid points, or to use externally generated tables over whose resolution and accuracy T-Spice has no control. Poorly chosen table parameters can result in inaccurate interpolation and degraded simulation results.

A simple example of the differences in results obtained from various table resolutions is shown in the figure below. The input file described a current mirror and used a Level 2 *n*-channel MOSFET model. The DC operating point was computed by direct evaluation to provide a standard against which the table-based results were compared. Each bar in the figure represents the percentage difference between the table-simulated result (at a given number of grid points) and the direct-evaluation standard — in every case, well under one-quarter of one percent.



Parametric Analysis

Under many circumstances, T-Spice will be required to study the effects on circuit performance of variations in parameter values. For example, parametric analysis can be used to evaluate multidimensional trends in the output over defined ranges of input values, or the sensitivity of circuit behavior to random fluctuations in fabrication conditions.

A large range of parameters may be systematically and automatically varied:

- External parameters (such as temperature)
- Simulation parameters (such as tolerances)
- Device parameters (such as input voltage level)
- Model parameters (such as transistor length)

Three types of parametric analysis are made possible by T-Spice: *parameter sweeping*, *Monte Carlo analysis*, and *optimization*. Discussions of T-Spice syntax and corresponding examples are included for all three analysis types in the chapter titled Parametric Analysis on page 486

In a *parameter sweep*, a specified parameter is held or initialized at a given value, on the basis of which all analyses requested by the input file are performed, and the results recorded. Then the parameter is incremented by a set amount, and the same analyses are repeated. This cycle is continued while the parameter is incremented through a defined range of values.

Parameter values may be swept *linearly* — in identical increments, typically through a limited range — or *logarithmically* — in exponential increments, typically through a range spanning multiple orders of magnitude.

An example illustrating T-Spice input for parameter sweeping is given in Parameter Sweeps on page 487.

Monte Carlo Analysis

Monte Carlo analysis generates "random" variations in parameter values by drawing them *probabalistically* from a defined distribution. For each value thus chosen, all analyses requested by the input file are performed, and the results recorded. Monte Carlo analysis is performed using the keyword **sweep**; syntax is described in **.step** (page 128).

Parameter values may be drawn from tunable *uniform*, *Gaussian*, or *random limit* distributions.

For a more detailed description of Monte Carlo analysis, see Monte Carlo Analysis on page 491.

If an aspect (or aspects) of the desired circuit performance can be specified quantitatively, T-Spice can search through a multidimensional "space" of parameters — that is, vary several parameters systematically and simultaneously — to determine the combination of parameter values that *optimizes* the specified performance measure (or set of measures).

Each run, using a particular combination of parameter values, produces a new value for each performance measure studied. Optimization is achieved by varying parameters in an attempt to minimize

$$\sum_i \left[w_i \frac{(G_i - R_i)}{G_i} \right]^2 \tag{0.3}$$

where G_i is the *goal* or desired value of the *i*th performance measure; R_i is the *result* or actual value of the *i*th performance measure, for a particular combination of parameters; and w_i is the *weight* or importance assigned to the *i*th performance measure relative to the other measures used. The quantity $w_i(G_i - R_i)/G_i$ is also called the *error*.

The choice of the *next* combination of parameters to test after each run, on the basis of the current total error, is the heart of the optimization algorithm. The algorithm employs *gradient descent*; that is, it attempts to find the steepest (fastest) "path" through the "space" of parameters that will lead to the minimum, by estimating the gradient in various directions.

For a description of T-Spice input needed to set up and invoke optimization, see Optimization on page 493. This section also includes a tutorial example illustrating optimization (see Example 3: Optimization on page 495).

Error Types and Correction

The highly compressed, text-based nature of the SPICE circuit description language, while rendering it efficient, portable, and flexible, also makes it prone to user errors of various kinds — errors that, while they will seldom crash the program, often result in wrong answers or problems with convergence.

T-Spice makes some corrections automatically, replacing improbable values with default ones or ignoring improper commands and specifications. In other cases, the Simulation Window shows the line number of the input file on which the error was found, or the name of the device whose specification T-Spice could not parse properly, and a brief description of the error. The most common errors can be divided into several categories.

Syntax Errors

- Unsupported commands or statements.
- Wrong spelling of commands, statements, or options.
- Failure to abide by conventions for names, comments, line continuation, numeric formats, unit abbreviations, or expressions.
- Wrong number or order of arguments on a simulation command or device statement.
- Numbers out of range.

Connectivity Errors

- Floating (unconnected) nodes.
- Nodes connected to only one device (except power supply and output nodes).
- Identical nodes referenced by different names.
- Different nodes referenced by identical names.
- Devices referred to that have not been previously defined.

Unless naming inconsistencies interfere with proper connectivity — producing floating nodes or devices with unconnected terminals — they will not be caught by T-Spice; it will simply assume that what is written is what is meant, and will produce misleading, implausible, or impossible results.

Convergence Errors

- Wrong metric prefix (order-of-magnitude error). For example, forgetting the **p** on a number intended to represent picofarads will not be caught by the syntax or connectivity checks but will probably lead to wildly wrong results.
- Ill-chosen tolerances.

To help prevent errors, use a *schematic export tool* to automate part of the process of writing a circuit description. These translate a *graphical* description of the circuit into a *textual* (SPICE-format) description. As long as the schematic has been drawn consistently (and this is easier to do graphically than textually), connections will be specified and nodes and devices named properly.

In addition, *use comments liberally*. Take advantage of the multiple ways by which comments can be indicated in a T-Spice input file to add structure and clarity to the circuit description. This will make later use of the file, whether for further study or for debugging, much simpler.

To deal with a *syntax error*, note the line number or device name in the input file at which T-Spice found the error, and check the syntax there.

To deal with a *connectivity error*, check the structure of the circuit, as described in the input file, carefully against the original schematic or plan. (This will be easier if T-Spice's own connectivity checker detected the error and provided the appropriate line number or device name.)

To deal with a *convergence error*, check the input file carefully, making sure that any metric prefixes used are plausible. Adjust the tolerances if necessary.

For information on comments, see Input Conventions on page 42. For information on syntax specifications, see Simulation Commands on page 50 and Device Statements on page 141.

4 Input Conventions

Introduction

When a simulation is run, the input file is interpreted by the T-Spice parser. For T-Spice to function efficiently and as compatibly as possible with other versions of SPICE, the parser must enforce a number of language conventions. This chapter summarizes those conventions.

For more information, see Simulation Commands on page 50 and Device Statements on page 141.

Names

All nodes and devices in the circuit must be identified uniquely by their *names*. Node and device names have the following features.

- Their length is unlimited (except by hardware constraints).
- They can include all characters except tabs, spaces, semicolons (**;**), single quotes (**'**), curly braces (**{}**), parentheses, forward slashes (**/**), and equal signs (**=**).
- The dollar sign (**$**) can appear in names, but cannot be a name by itself.
- They are *case sensitive*. For example, the name **VDD** is different from **Vdd** and **vdd**.

Model names cannot start with digits.

The following examples are all valid names:

```
in            Alpha16              ONE[21]3_72
```

Reserved Names

T-Spice uses *reserved* node names for the default system ground: **gnd**, **GND**, **Gnd**, and **0** (zero). All instances of these nodes are connected and treated as the same node, which is fixed at a potential of 0.0 volts.

The following *keywords* (in any combination of uppercase and lowercase letters) cannot be used as names:

```
ac            bit           biti          bus
busi          dc            exp           inoise
off           onoise        params        pie
piei          pwl           pwlfile       poly
pulse         r             repeat        round
rounding      sffm          sin           sini
transfer
```

Device Names

Most device statements are of the form

```
Zxxx ...
```

where the variable ***Z*** represents the required key letter which uniquely specifies the device type, and the variable ***xxx*** represents a user-supplied alphanumeric string.

For example, MOSFET statements have a form following this example:

```
mtran1 d g s b nmos l=2um w=2um
```

In this example, **m** is the required key letter and **tran1** is the user-supplied string. The device’s name is **mtran1** (not **tran1**).

A particular device may be indicated in the input file by *either* case, for example, **m** or **M**, of the key letter — but the case must remain constant for the same device throughout the file.

Hierarchical Names

Hierarchical node names are used to refer to nested subcircuit nodes. Each level in the name of a node is separated by a period (**.**).

For example, the internal node **ing** in the **xnand** subcircuit contained in the **xadder** subcircuit is specified as

```
xadder.xnand.ing
```

Subcircuit Pin Name Aliasing

T-Spice recognizes subcircuit pin node names by their internal names as well as their global names.

For example, the netlist

```
.subckt test a b
r1 a c 100
r2 c b 100
.ends
x1 a1 b1 test
```

creates three nodes: **a1**, **b1**, and **x1.c**. **a1** and **b1** are globally recognized node names; but they may also be referred to by aliases **x1.a** and **x1.b**, respectively —for example, within a command such as **.print dc v(x1.a)**.

Comments

Comments provide information about the circuit, but are not processed as part of the formal circuit description. Comments are generally indicated by the presence of special characters called *delimiters*.

However, T-Spice *always treats the first line of the input file as a comment, even without comment delimiters*.

Other comments may be placed anywhere in the circuit description.

Several comment styles are allowed, for compatibility with other versions of SPICE:

- An asterisk (*****), dollar sign (**$**), or semicolon (**;**) in the first column of a line indicates that the entire line is a comment.
- A dollar sign or semicolon, but *not* an asterisk, anywhere in a line *other* than in the first column indicates that the rest of the line is a comment.
- C-style comments, enclosed by the delimiters **/*** and ***/**, can be used anywhere, except in the middle of multi-word commands (such as **.print tran**) or arguments. A C-style comment is not restricted to one line.

The comments in the following examples are highlighted:

```
* Lines beginning with asterisks,
$ or dollar signs,
; or semicolons are ignored.
r1 node1 node2 4k $ This comment can follow a command
c2 node3 node4 100f ; This is like a '$' comment
v1 node5 GND /* This is a
       C-style comment */ 3volt
```

A C-style comment that interrupts a command or argument may cause an error message. However, comments may appear *between* arguments.

The first two lines in the following examples would cause error messages:

```
.print /* wrong */ tran v(1)
.options prtdel /* wrong */ = 0.01
.options abstol=1e-8 /* OK */ reltol=1e-4
```

Line Continuation

Input file lines may be of any length; however, it is often convenient to break up long lines for readability.

A plus sign (**+**) *in the first column* denotes line continuation. For example:

```
.model nmes NMF
+ vto=-2.5 rs=100 rd=200 pb=0.7
+ alpha=2.0 cgs=500.0f cgd=100.0f
```

Comments in Continued Lines

Comments may appear between continued lines. However, blank lines may *not* appear between continued lines.

A continued line may include a dollar sign (**$**) or semicolon (**;**) comment symbol, in which case the rest of the line is ignored.

No plus sign is needed for a comment line, a line continuing a C-style comment, or a line continuing an expression (see Parameters on page 46).

For example:

```
.options
* now we declare several options, continuing the line
+ prtdel = 10ms $ (; would work too)
+ abstol = 1e-10 /* we can put a
       C-style comment here */
```

Expressions and Continued Lines

Plus signs *cannot be used to indicate addition in the first column* within multi-line expressions.

For example:

```
.options numnd = '7 +
log(10.1)     $ will be added to previous term
+ 2'          $ will NOT be added - ‘+’ is ignored
```

Numbers and Units

Some commands and statements require arguments representing physical quantities with attached units (such as seconds or volts). Such numbers can be expressed in floating point, scientific, or fixed-point notation, and can be followed by metric abbreviations indicating order of magnitude.

The base units (**s**, **v**, **a**, **f**, **h**) are implicit from the context and are optional.

For example, the following expressions can all specify 12 nanoseconds in the appropriate context:

```
12ns          12e-9         .012u         12000p
```

Acceptable metric abbreviations are as follows.

Abbreviation	***Prefix***	***Meaning***
t or **T**	tera-	10^{12}
g or **G**	giga-	10^{9}
meg or **MEG**	mega-	10^{6}
x or **X**	mega-	10^{6}
k or **K**	kilo-	10^{3}
m or **M**	milli-	10^{-3}
‘**u** or **U**	micro-	10^{-6}
n or **N**	nano-	10^{-9}
p or **P**	pico-	10^{-12}
f or **F**	femto-	10^{-15}

The abbreviation **f** is ambiguous because it can mean either the scale indicator “femto-” or the unit “farad.” T-Spice employs the following convention: **f** by itself means “femto-.” Thus **100f** means “100 femto-," where the unit is clear from the context. Where **f** precedes a base unit, as in **ff** and **fs**, or follows a metric abbreviation, as in **uf**, there is no ambiguity. When "farad" by itself is meant, no unit should be used.

A commonly used unit abbreviation is **mil** (or **MIL**), representing 10^{-3} inch.

Parameters

You can declare parameters and assign them values with the **.param** command. Parameters cannot be reassigned values within an input file.

Parameter names can contain any characters except tabs, spaces, commas, curly braces, parentheses, single quotes, square brackets, equal signs, and algebraic operators (**+ – * / ^**).

Expressions

Any number in a command or statement may be replaced by an algebraic *expression*. Expressions must conform to the following conventions.

- They must be enclosed by single quotes (').
- They may span several lines. The plus sign (**+**) is *ignored* if it appears in the first column of a continued expression.
- They may include comments that do not interrupt numeric values or parameter names.

Expressions may involve any valid combination of numbers, parameters (**.param** (page 102)), operations, algebraic functions, and user-defined functions. T-Spice evaluates expressions according to a standard mathematical operator precedence, shown below. Level 1 has the highest operator precedence, and level 10has the lowest.

Priority	***Operation***	***Description***
Level 1	**(*x*)**	Parentheses
Level 2	***f(x)***	Function call
Level 3	***x^y***	Exponentiation
Level 4	–***x***	Unary negation
Level 5	***x***×***y*** ***x/y***	Multiplication Division
Level 6	***x***+***y*** ***x***–***y***	Addition Subtraction
Level 7	<, <=, >, >=	Relational operators
Level 8	==, !=	equality and inequality
Level 9	x && y	logical AND
Level 10	x || y	logical OR

The operations and functions available in T-Spice are summarized in the following tables (in which the variables ***x*** and ***y*** represent numbers, parameter names, or subexpressions). All angles are in radians.

Operation	***Description***
(*x*)	Parentheses — to override operator precedence
x + y	Addition
x – y	Subtraction
x * y	Multiplication
x / y	Division
x^y, x**y	Exponentiation (x^y)
–x	Unary negation
x < y, x <= y, x > y, x >= y	Relational operators - return 1 if the relation is true, otherwise 0
x == y, x != y	Equality operators - return 1 if the (in)equality is true, otherwise 0
x && y	Logical AND - returns 1 if x and y are true (non-zero), otherwise 0
x || y	Logical OR - returns 1 if x or y is true (non-zero), otherwise 0

Built-in Functions

Function	***Description***
abs(*x*)	Absolute value of *x* (same as **fabs**)
acos(*x*)	inverse cosine of *x* (error if |*x*| > 1)
asin(*x*)	Inverse sine of *x* (domain error if |*x*| > 1)
atan(*x*)	Inverse tangent of *x* (range: [–π / 2, π / 2])
atan2(*x*,*y*)	Inverse tangent of *y/x* (range: [–π, π])
ceil(*x*)	Smallest integer not less than *x*
cos(*x*)	Cosine of *x*
cosh(*x*)	Hyperbolic cosine of *x*
db(***x***)	*x* in decibels: (sign of *x*)·20·$\log_{10}$(|*x*|)
err(*x*,*y*)	error analysis; abs(*x*-*y*) / max(*x*,*y*)
exp(*x*)	e^x
fabs(*x*)	Absolute value of *x* (same as **abs**)
floor(*x*)	Largest integer not greater than *x*
fmod(*x*,*y*)	Remainder of *x/y* (error if *y* = 0)
if(*c*,*a*,*b*) or **c ? a : b**	conditional statement, if ***c*** is true, then ***a***, else ***b.*** Two syntax variations are supported for the conditional statement - the **if()** function call, and the C style of conditional expression
int(***x***)	Convert *x* to an integer, removing the fractional portion

Function	***Description***
ldexp(*x,y*)	$x \times 2y$ for integer y
log(*x*)	Natural logarithm of x (error if $x \leq 0$)
log10(*x*)	Logarithm (base 10) of x (error if $x \leq 0$)
log2(*x*)	Logarithm (base 2) of x (error if $x \leq 0$)
max(***x1***, ***x2***)	Evaluates to the maximum of the two arguments.
min(*x1*, ***x2***)	Evaluates to the minimum of the two arguments.
pow(*x,y*)	Exponentiation (x^y)
pwr(***x,y***)	In HSPICE compatibility mode (default): (sign of x)·$\|x\|^y$ In PSPICE compatibility mode (.option spice=pspice) exponentiation (x^y), equivalent to **pow(x,y)**
pwrs(***x,y***)	signed power function, (sign of x)·$\|x\|^y$
sgn(*x*)	Sign of x: -1 if $x < 0$ 0 if $x = 0$ 1 if $x > 0$
sign(***x***,***y***)	(sign of y)·$\|x\|$
sin(*x*)	Sine of x
sinh(*x*)	Hyperbolic sine of x
sqrt(*x*)	Square root of x (-sqrt($\|x\|$) if $x < 0$)
stp(expression) or **stp**(expression1, expression2)	The first syntax evaluates to 0 if expression is negative, and 1 otherwise. The second syntax evaluates to 0 if *expression1* is less than -*expression2*, to 1 if *expression1* is greater than *expression2*, and to *(expression1+expression2)/(2*expression2)* otherwise. The second syntax thus provides a continuous approximation to a step function.
table(x, x1, y1, x2, y2, … , xn, yn)	All arguments of the table function may themselves be subexpressions. The table function evaluates the piecewise linear function defined by the points *x1*, *y1*, *x2*, *y2*, … , *xn*, *yn*, connected by straight lines. The function value is the *y*-value of that function at *x*. The points are automatically sorted in ascending order of *x* values to form the piecewise linear function. If *x* is less than the smallest x_k, then the return value is the *y*-value corresponding to the smallest x_k. Similarly, if *x* is greater than the largest x_k, then the return value is the *y*-value corresponding to the largest x_k.
tan(*x*)	Tangent of x
tanh(*x*)	Hyperbolic tangent of x

Differentiation and Integration functions

In addition to the Standard math library functions, T-Spice provides a set of functions for computing integrals and derivatives of data.

The **ddt(*f*), d2dt(*f*),** and **idt(*f*) (***aka* **sdt(*f*))** functions compute the time derivatives and integrals of transient simuation data.

The **ddx(*f,x*), d2dx(*f,x*),** and **idx(*f,x*) (*aka* sdx(*f,x*))** functions compute the derivatives and integrals of the variable or expression ***f*** with respect to the independent variable or expression ***x***.

The time-based integration and differentiation functions, and the generic ***x*** dependent forms of the functions, both use polynomial fits to the data for computing the integrals and derivatives. In the case of the transient functions, the order of the polynomial tracks that of the transient simulation engine, which is controled by the **maxord** (page 234) option. The generic ***x*** dependent functions use a second-order polynomial fit, by default.

The order of the function equations can be changed using the **const_dt_maxord**, and **const_dx_maxord** options:

```
.options const_dt_maxord=[0-4] ;default value is 0, selecting transient order
.options const_dx_maxord=[1-4] ;default value is 2
```

Note: The quality of integrals is highly dependent upon the order of the fitting polynomial. It is important to select an order of equations which is an appropriate fit to the data. If the dependent function ***f*** is highly irregular or is a square-shaped digital signal, then a 1st or 2nd order integration is best. Higher order equations should only be used for very smooth data.

If the integration solutions are suspicious, perhaps containing very large integral values, then the solutions should be verified by re-running the simulation with the function integral order set to 1, which results in a piecewise summation of integrands, without higher order effects.

Function	*Description*
ddt(*f*)	Derivative of ***f*** with respect to time
d2dt(*f*)	Second derivative of ***f*** with respect to time
idt(*f*) | sdt(*f*)	Integrate ***f*** in time
ddx(*f*,*x*)	Derivative of ***f*** with respect to ***x***
d2dx(*f*,*x*)	Second derivative of ***f*** with respect to ***x***
idx(*f*,*x*) | sdx(*f*,*x*)	Integrate ***f*** in ***x***

5 Simulation Commands

Introduction

This chapter describes features, outlines syntax, and gives examples for the *simulation commands* of the T-Spice circuit description language.

The commands are listed in alphabetical order. Many commands have *options*, which branch to different modes, and *arguments*, which indicate expressions, nodes, or devices to be operated on. Options and arguments must be separated by spaces or new lines (with line continuation).

The *Syntax* sections follow these conventions:

- ***Italics*** indicate variables to be replaced by actual names, numbers, or expressions.
- Curly braces **{ }** indicate alternative values for the same option or argument.
- Square brackets **[ ]** enclose items that are *not required*.
- Vertical bars **|** separate alternative values for the same option or argument.
- Ellipses **...** indicate items that may be repeated as many times as needed.

Brackets, vertical bars, and ellipses are *not* typed in the input file. All other characters are typed as shown. For further information, see: Input Conventions on page 42; Device Statements on page 141.

.ac

Performs AC analysis to characterize the circuit's dependence on small-signal input frequency: the DC operating point is computed, a linearized small-signal model is constructed at the DC operating point, and the circuit's response over a range of frequencies is measured.

- Small-signal parameters are reported to the main simulation output file. You can disable this reporting or specify a different output file using the command **.acmodel** (page 54). Reporting is automatically disabled if the simulation contains more than 1000 nodes.
- AC analysis results can be reported with the commands **.print ac**, **.probe ac** (for binary output), or **.measure ac**. For additional information on these commands, see **.print** (page 108)**, .probe** (page 122)**,** and **.macro /.eom** (page 77).
- The frequency can be varied linearly, by octaves, or by decades, by specifying a total number of test points, or by listing specific test frequencies.

Syntax

```
.ac {lin|oct|dec} num start stop [sweep swinfo] [analysisname=name]
```

or

```
.ac list f1 f2 ...fn [sweep swinfo] [analysisname=name]
```

or

```
.ac poi num f1 f2 ...fn [sweep swinfo] [analysisname=name]
```

Parameters

lin | **oct** | **dec**

Frequency variation mode.

- **lin**: linear sweep.
- **oct**: logarithmic sweep by octaves.
- **dec**: logarithmic sweep by decades.

list

Using the **list** keyword with **.ac** allows the user to specify a list of frequencies (***values***) for which the analysis is to be performed.

poi

Using the **poi** keyword with **.ac** allows the user to specify a list of frequencies (***values***) for which the analysis is to be performed. The **poi** mode of processing is the same as the **list** mode, except that the syntax for **poi** requires that the number of frequencies, ***N***, be specified next.

num

Frequency count.

- Linear mode: Number of frequencies between start and stop.
- Octave mode: Number of frequencies per octave between start and stop.
- Decade mode: Number of frequencies per decade between *start* and *stop*.
- **poi** mode: Total number of frequency values.

start

First frequency. (Unit: Hertz.)

stop

Last frequency. (Unit: Hertz.)

sweep

Specifies the parameter values of the sweep for which analysis will be performed. The **sweep** keyword is equivalent to **.step** (page 128) and uses the same parameter syntax. However, **sweep** applies only to one analysis command, while **.step** applies to all analysis commands in the input file. If **sweep** is specified on an analysis command and **.step** is present, the **sweep** sweep is nested inside the **.step** sweep. The parameter **sweep** may be used to specify a parametric sweep, Monte Carlo analysis, or optimization.

analysisname

Specifies an analysis name that will be referenced by an **.optimize** simulation command. For further information, see **.optimize** (page 93).

Examples

```
.ac DEC 5 1MEG 100MEG
```

Defines a frequency sweep from 1 MHz to 100 MHz by decades, with 5 points per decade.

```
.ac LIN 100 10K 100MEG
```

Defines a linear frequency sweep from 10 kHz to 100 MHz with 100 equally spaced points.

```
.ac list 5 50 500 5000 sweep rval dec 10 1 1000
```

Performs a logarithmic sweep of the parameter **rval** from 1 to 1000, using 10 points per decade; also performs an AC analysis at the four specified frequencies for each value of **rval**.

.acmodel

Modifies reporting of small-signal model parameters and operating points for specified devices in conjunction with AC analysis or DC operating point analysis. Small-signal parameters are automatically reported to the main simulation output file with the use of either **.ac** or **.op** in the input file.

- If **.ac** (page 51) and **.op** (page 91) are missing from the input file, **.acmodel** is ignored.
- No small-signal data is reported if the simulation model has more than 1000 nodes.
- Small-signal data are available for diodes, resistors, BJTs, JFETs, MESFETs, MOSFETs, and devices modeled with user-defined external models.

Syntax

```
.acmodel [output file ] { [device [[,] device ...]]}
```

output file Output filename. If no filename is specified, then the output will be reported to the main simulation output file.

device Device(s) for which small-signal model parameters and operating points are to be given. If no devices are specified, then no small-signal data is reported.

Examples

```
.acmodel {mt1 mt2}
```

Prints data for devices **mt1** and **mt2**.

```
.acmodel {}
```

Turns off all small-signal parameter reporting.

.alter

Causes a simulation to be repeated with slight changes as specified after the **.alter** command. Multiple **.alter** commands can appear in the same netlist. T-Spice performs the first simulation with all commands that occur before the first **.alter** command; the second simulation incorporates changes between the first and second **.alter** commands; the third simulation incorporates changes between the second and third **.alter** commands, and so on. The optional ***altername*** string identifies the **.alter** block that follows and is used to identify the corresponding simulation output in output files.

A **.alter** block can contain any legal T-Spice statement. This command occurs at the end of a complete input file.

- Elements, option values, parameter values, and **.connect** (page 57) and **.model** (page 85) statements in a **.alter** block replace equivalent statements in the main netlist if the name of the element, parameter, option, data set, or model parameter set matches. If no name match can be made, the statement in the **.alter** block is simply added to the simulation.
- Statements are cumulative and progressive from one **.alter** block to the next such that any additions or changes made in block ***N*** will also occur (unless superseded) in block ***N+1***.
- Initial conditions (**.ic** (page 70)) and **.nodeset** (page 89) replace equivalent commands that operate on the same nodes or devices. If no equivalent command is found in the main netlist, the command is simply added.
- If a **.alter** block contains a **.temp** (page 135) command, any **.temp** commands in the original netlist are replaced with the new **.temp** command.
- The commands **.lib** (page 74) and **.if ... / .elseif ... / .else / .endif** (page 71) can be used within **.alter** (page 55) blocks to include command and element statements.
- The **.del lib** (page 61) command can be used to delete a library section included in the original input file. All commands and elements in the library section are ignored during the “altered” simulation.
- Simulation commands such as **.ac** (page 51), **.tran** (page 137), and **.step** (page 128) do not replace commands in the original input file, but are simply added on as new commands.

Syntax

.alter [*altername*]

Examples

```
v1 1 0 1
r1 1 2 1k
r2 2 0 1k
.op
.alter
r1 1 2 2k
.alter r2_4k
r2 2 0 4k
.end
```

This performs three simulations:

The first uses **r1=1k**, **r2=1k**, and is identified by **alter=0**; the output is **v(2)=0.5**.

The second uses **r1=2k**, **r2=1k**, and is identified by **alter=1**; the output is **v(2)=0.333333 (1/3)**.

The third uses **r1=2k**, **r2=4k**, and is identified by **alter=r2_4k**; the output is **v(2)=0.666667 (2/3)**.

The abbreviated output would be:

```
*SEDIT: Alter=0
*SEDIT: Analysis types DCOP 1 ACMODEL 0 AC 0 TRANSIENT 0 TRANSFER 0 NOISE 0

* BEGIN NON-GRAPHICAL DATA

DC ANALYSIS - alter=0
v(2) =   5.0000e-001
v(1) =   1.0000e+000
i(v1) =  -5.0000e-004

* END NON-GRAPHICAL DATA

*SEDIT: Alter=1
*SEDIT: Analysis types DCOP 1 ACMODEL 0 AC 0 TRANSIENT 0 TRANSFER 0 NOISE 0

* BEGIN NON-GRAPHICAL DATA

DC ANALYSIS - alter=1
v(2) =   3.3333e-001
v(1) =   1.0000e+000
i(v1) =  -3.3333e-004

* END NON-GRAPHICAL DATA

*SEDIT: Alter=2
*SEDIT: Analysis types DCOP 1 ACMODEL 0 AC 0 TRANSIENT 0 TRANSFER 0 NOISE 0

* BEGIN NON-GRAPHICAL DATA

DC ANALYSIS - alter=r2_4k
v(2) =   6.6667e-001
v(1) =   1.0000e+000
i(v1) =  -1.6667e-004

* END NON-GRAPHICAL DATA
```

.connect

This command will connect two nodes in your circuit, so that the two nodes will be simulated as only one node. In essence, one node name becomes an alias for another node name.

Both nodes must be at the same level in the circuit design that you are simulating: you cannot connect nodes that belong to different subcircuits

Syntax

.connect *node1 node2*

If you connect node2 to node1, you can then refer to either node1 or node2 in other simulation commands, and it will refer to the same node.

Examples

```
vcc 0 cc 5v
r1 0 1 5k
r2 1 cc 5k
.tran 1n 10n
.print i(vcc) v(1)
.alter
.connect cc 1
.end
```

The first .tran simulation includes two resistors. Later simulations have only one resistor, because r2 is shorted by connecting cc with 1.

You may also use multiple .connect statements to connect several nodes together:

```
.connect node1 node2
.connect node2 node3
```

connects both node2 and node3 to node1. The T-Spice simulation evaluates node voltages and related terms only for node1; node2, and node3 are the same node as node1.

Note: If you set **.option node**, T-Spice prints out a node connection table.

.data

Used to incorporate external numerical data into simulations. The data can be used to specify reference data for error measurements, which is often used in conjunction with optimization for model parameter extraction. Also used to specify parametric sweeps in which several variables are swept simultaneously.

Syntax

```
.data dataname
+ colname [colname [...]]
+ value [value [...]]
+ value [value [...]]
...
.enddata
```

The number of columns is equal to the number of values per row.

dataname	Name assigned to the data set.
colname	Name assigned to a column of data within the data set.
value	Numeric parameter value. These values may be expressions, but must not depend on other parameter values.

A data set can be used with the **.macro /.eom** (page 77) command to compute differences between simulation output curves and corresponding external data.

A data set can also be used to define a parameter sweep on an **.ac** (page 51), **.dc** (page 60), **.step** (page 128), or **.tran** (page 137) command.

The syntax for the ***sweep*** parameter on those commands is as follows:

```
data=dataname
```

where ***dataname*** refers to a **.data** statement of the same name. Each row of numbers in the **.data** statement corresponds to a sweep step, and each column refers to a sweep parameter (which must be a global parameter defined using **.param**) and the values which the parameter takes on. The ***colname*** strings identify the parameters to be swept. For each sweep step, the sweep parameters take on the values listed in a row of data.

Examples

```
.data idsdata
+ time              v1dat
+ 0                 0
+ 1u                0
+ 1.1u              5
+ 2u                5
.enddata
.tran 0.1u 2u
.measure tran v1fit err1 v1dat v(1)
```

This example computes the difference between the externally supplied data curve **v1dat** and the simulated transient analysis response voltage **v(1)**.

```
.data idsdata
+ vds          ids
+ 0.0000e+00   0
+ 5.0000e-01   4.8565e-10
+ 1.0000e+00   2.2752e-06
+ 1.5000e+00   1.2558e-05
+ 2.0000e+00   3.1509e-05
+ 2.5000e+00   5.9462e-05
+ 3.0000e+00   9.6027e-05
+ 3.5000e+00   1.4135e-04
+ 4.0000e+00   1.9569e-04
+ 4.5000e+00   2.5978e-04
+ 5.0000e+00   3.2942e-04
.enddata
.param vds=0 ids=0
vd drain source vds
m1 drain gate source bulk nmos 1=2u w=2u
.dc data=idsdata
.measure dc idsfit err1 ids id(m1)
```

This example computes the (relative) difference between the externally supplied data curve **ids** and the simulated MOSFET drain current **id(m1)**. This is particularly useful in conjunction with optimization for model parameter extraction, where the purpose of the simulation is to select model parameter values which yield the best match between simulated and measured data curves.

.dc

Performs DC transfer analysis to study the voltage or current at one set of points in a circuit as a function of the voltage or current at another set of points. Can also be used for linear or logarithmic sweeps of DC voltage or current; for sweeps of parameters other than voltage and current source values; and for Monte Carlo analysis or optimization.

- Transfer analysis is done by *sweeping* the source variables over specified ranges, and recording the output.
- Up to three parameters can be specified per **.dc** command.
- When two or more sources are specified, the last-named source "controls" the sweeping process (see Examples, below).
- The specified current or voltage sources must exist—that is, be defined by **i** or **v** device statements elsewhere in the input file.
- DC transfer analysis results can be reported with the **.print** (page 108) **dc, .probe** (page 122) **dc**, and **.macro /.eom** (page 77) **dc** commands.

Syntax

```
.dc swinfo [[sweep] swinfo [[sweep] swinfo]]
```

Refer to **.step** (page 128) for a syntax description of ***swinfo***.

Examples

```
.dc isrc 0 1e-6 0.1e-6
```

Current source **isrc** is swept from 0 to 1 microampere in 0.1-microampere steps.

```
.dc vin 0 5 0.05 VCC 4 6 0.5
```

Names two voltage sources: **vin**, to be swept from 0 to 5 volts in 0.05-volt steps, and **VCC**, to be swept from 4 to 6 volts in 0.5-volt steps. The second source "controls" the sweep: **VCC** is initially set to 4 volts, while **vin** is swept over its specified range. Then **VCC** is incremented to 4.5 volts, and **vin** is again swept over its range. This process is repeated until **VCC** reaches the upper limit of its specified range.

```
.data sweep_params
+ vds          r2           length
+ 0.0          1k           10u
+ 1.0          500          12u
+ 2.0          100          14u
.enddata
.dc data=sweep_params
```

This example performs a DC sweep defined using a **.data** statement. There are three sweep steps, and the parameters **vds**, **r2**, and **length** are varied in the sweep. On the first sweep step, **vds=0**, **r2=1000**, and **length=10u**; on the second sweep step, **vds=1**, **r2=500**, and **length=12u**; on the third sweep step, **vds=2**, **r2=100**, and **length=14u**.

.del lib

Used to delete a section from a library file previously included using the **.lib** (page 74) command. The **.del lib** command is used in **.alter** (page 55) blocks, typically to replace a library section with a different one.

Syntax

```
.del lib filename section
```

filename Name of the referenced library file. If the referenced filename or path contains a space, enclose the entire path in single or double quotation marks.

section Name of the library section in ***filename*** that is to be deleted.

Examples

```
.lib bsim3model.md typical
.alter
.del lib bsim3model.md typical
.lib bsim3model.md fast
.alter
.del lib bsim3model.md fast
.lib bsim3model.md slow
.end
```

T-Spice input such as shown above might be used to run a simulation three times, first with **typical**, then with **fast**, and finally with **slow** model parameters. First, the library section called **typical** is loaded for the first simulation. The second simulation incorporates changes between the first and second **.alter** commands, so that the **typical** library section is deleted and replaced with the **fast** library section. Similarly, the third simulation replaces the **fast** library section with the **slow** one.

.end

Signifies the end of the circuit description.

- Any text in the input file after the **.end** command is ignored.
- The **.end** command is optional in T-Spice but is included for compatibility with generic SPICE.

Syntax

```
.end [comment]
```

.enddata

Signifies the end of a **.data** statement.

- The **.enddata** command *must* accompany a **.data** (page 58) command.

Syntax

```
.enddata [comment]
```

.endl

Signifies the end of a library definition.

- The **.endl** command *must* accompany a **.lib** (page 74) command.

Syntax

```
.endl [comment]
```

.ends

Signifies the end of a subcircuit definition.

- The **.ends** command *must* accompany a **.subckt** (page 132) command.

Syntax

```
.ends [comment]
```

.four

Performs Fourier analysis on transient analysis data.

- Fourier components (magnitude and phase) are computed for a given fundamental frequency and corresponding to a specified number of integer multiples of the fundamental frequency.
- The DC Fourier component is computed, as well as the total harmonic distortion, defined as

$$\frac{1}{R_1} \cdot \left(\sum_{m=2}^{nfreqs} R_{m^2} \right) \tag{0.4}$$

  where *Rm* is the magnitude of the *m*th Fourier component.

- The **.four** command is ignored if no **.tran** command is found.

Syntax

```
.four F list [nfreqs=N] [npoints=P] [interpolate=I]
```

F

Fundamental frequency.

list

Output variables for which the analysis is to be performed. Each of these can be any valid output item from a **.print** (page 108) **tran** command, including output expressions.

N

Number of frequencies for which Fourier components are determined. (Default: 9.)

P

Number of points over which transient analysis data is interpolated to fit. These points equally divide the analysis interval (T–J,T), where T is the final time specified on the corresponding **.tran** (page 137) command, and $J = 1/\boldsymbol{F}$. Increasing ***P*** improves accuracy but increases simulation time and memory use. (Default: 100.)

I

If 0, T-Spice inserts an actual computed time point at each place where a Fourier analysis time point is needed without interpolating transient data to fit on *np*. If 1, Fourier analysis is based on interpolated data. (Default: 1.)

Examples

```
v1 2 0 sin (4 10 9e6 0 0 20)
v2 1 2 sin (0 3 3e6 0 0 -50)
.tran 1u 10u
.four 3e6 v(1) npoints=1000
```

The formula for v(1) is:

$$\begin{aligned} &v(1) = 4 + \\ &10\sin[2\pi(9 \times 10^6 t + 20/360)] + \\ &3\sin[2\pi(3 \times 10^6 t - 50/360)] \end{aligned} \tag{0.5}$$

The analytic Fourier response for this formula is as follows:

- The DC component is 4.
- The first harmonic (at a frequency of 3 MHz) has a magnitude of 3 and a phase of -50 degrees.
- The third harmonic (at a frequency of 9 MHz) has a magnitude of 10 and a phase of 20 degrees.
- All other harmonics have a zero Fourier component.

The output will be:

```
* BEGIN NON-GRAPHICAL DATA
FOURIER ANALYSIS RESULTS

Fourier components of transient response  v(1)

DC component =   4.0011e+000

Harmonic no   Frequency<Hz>    Fourier comp    Normalized FC    Phase<deg>  Normalized phase
          1     3.0000e+006     2.9989e+000     1.0000e+000  -4.9651e+001       0.0000e+000
          2     6.0000e+006     2.4224e-002     8.0777e-003   3.5132e+001       8.4783e+001
          3     9.0000e+006     9.9949e+000     3.3328e+000   2.0564e+001       7.0215e+001
          4     1.2000e+007     3.4401e-002     1.1471e-002  -1.6494e+002      -1.1529e+002
          5     1.5000e+007     1.8803e-002     6.2701e-003  -1.6786e+002      -1.1821e+002
          6     1.8000e+007     1.3374e-002     4.4595e-003  -1.6966e+002      -1.2001e+002
          7     2.1000e+007     1.0534e-002     3.5126e-003  -1.7085e+002      -1.2120e+002
          8     2.4000e+007     8.7570e-003     2.9201e-003  -1.7169e+002      -1.2204e+002
          9     2.7000e+007     7.5266e-003     2.5098e-003  -1.7229e+002      -1.2264e+002

Total harmonic distortion =         333.3 percent

* END NON-GRAPHICAL DATA
```

where the DC component is 4.0011e+000, the first harmonic has a magnitude of 2.9989 and a phase of -4.9651e+001, the third harmonic has a magnitude of 9.9949e+000 and a phase of 2.0564e+.001.

Note that harmonics one and three have the exponent e+000, while the magnitude of the other harmonics—e-002 or even e-003—is quite small in comparison.

.global

Specifies nodes with global scope.

- Global node names refer to the *same* nodes both inside and outside subcircuit definitions.
- Ground (**0**, **gnd**, **Gnd**, or **GND**) is automatically defined to be a global node.

Syntax

```
.global node1 [[,] node2 ...]
```

.gridsize

Specifies the number of gridpoints contained in each dimension of a device type's internal tables. (See Evaluation Methods on page 37, and also .vrange on page 140 regarding internal tables.)

- Charge and current tables for a given device type have the same gridsize.
- All devices of the same type within a simulation have the same gridsize.
- Increasing gridsize tends to increase time and memory requirements as well as accuracy.

Syntax

```
.gridsize type A [B [C]]
```

type	Device type (see below).
A B C	Numbers of gridpoints in all table dimensions (see below). All dimensions for a given device must be included. Extra values are ignored.

Each device type has a certain *number* of dimensions, an *order* in which the dimensions must be listed, and a *default* gridsize if the **.gridsize** command is not given for that type. The possible values of ***type***, with the corresponding dimension orders and default number of gridpoints (in parentheses), are as follows:

Device	***type***	***A***	***B***	***C***
MOSFET	**mos**	*Vds* (64)	*Vgs* (128)	*Vbs* (10)
MESFET	**mes**	*Vds* (30)	*Vgs* (60)	*Vbs* (6)
JFET	**jfet**	*Vds* (20)	*Vgs* (20)	—
Diode	**diode**	*Vpn* (35)	—	—

.ic

Sets node voltages or inductor currents for the duration of a DC operating point calculation.

- DC operating points are calculated by the **.ac** (page 51), **.dc** (page 60), **.op** (page 91), **.tf** (page 136), and **.tran** (page 137) commands.
- The **.ic** command adds a voltage source present only in DC (not transient) simulations.
- The specified nodes are allowed to float if a transient analysis is subsequently requested in the input file. For further information on transient analysis, see **.tran** (page 137).
- Nodes and devices within subcircuits can be accessed with hierarchical notation in the form **x*instance*.x*instance*.*node***.
- To set initial guesses for node voltages, use the **.nodeset** (page 89) command.
- **.ic** commands within subcircuit definition (**.subckt** (page 132)/**.ends** (page 65)) blocks are replicated for each subcircuit instance.

Syntax

```
.ic node=X [[,] node=X ...]
.ic v(node [,node])=X [[,] v(node [,node])=X ...]
.ic i(inductor)=X [[,] i(inductor)=X ...]
```

node	Node whose voltage is to be initialized. (Default reference node: ground.)
inductor	Inductor whose current is to be initialized.
X	Node-to-node or node-to-ground voltage or inductor current value. (Unit: volts or amperes.)

Examples

```
.ic a=5, b=5, c=5
```

Assigns initial voltages of 5 volts (relative to ground) to nodes **a**, **b**, and **c**.

```
.ic v(a,b)=5
```

Sets the initial voltage *between* nodes **a** and **b** to 5.

.if ... / .elseif ... / .else / .endif

The **.if, .elseif, .else, and .endif** conditional statements may be used in a netlist to control which simulation commands, device statements, and device models will be included in the simulation.

The .if and .elseif commands have required conditional statements which will be evaluated to either true or false (non-zero or zero). Processing will then proceed, according to the conditional value. The first condition block which evaluates to true will be the selected block, or the .else block will be evaluated if no other blocks are true.

- Conditional statements may be nested.
- You can have an unlimited number of **.elseif** statements in your conditional.
- Parameter assignments that are contained within a condition's statement block do not effect the condition evaluation.
- Statements that are part of a condition statement block are only evaluated if and when the containing condition statement is evaluated to true.

Syntax

```
.if condition1
       < statement block1 >
[ .elseif condition2 ]
       < statement block2 >
[ .elseif condition3 ]
       < statement block3 >
[ .elseif ... ]
       ...
[ .else ]
       < statement blockn >
.endif
```

condition A value or an expression, enclosed in parentheses or quotes, which will be evaluated to control the selection of one of the statement blocks.
Any non-zero expression value will be considered **true**, and a zero value is **false**.

statement block Any valid T-Spice statements

Examples

The simplest example of a conditional statement has a single conditional **.if** block:

```
.param debug=1
.if (debug)
       .options echo=1
       .options verbose=2
       .options list nomod=0 node
.endif
```

An example which demonstrates all types of conditional statements:

```
.if (technology==49 && fast)
       .lib mos49.md FF
.if (technology==49 && fast==0)
       .lib mos49.md TT
.elseif (technology==53)
       .lib mos53.md TT
.else
       .lib mos2.md TT
.endif
```

.include

Includes the contents of the specified file in the input file.

- The **.include** command can be nested (included files can include other files, and so on) as deeply as hardware and operating system constraints permit.

Syntax

```
.include filename
```

filename The file to be included. It must exist in the current directory or in the T-Spice search path. Absolute or relative path names (according to the conventions of the operating system) can be used. If the referenced filename or path contains a space, enclose the entire path in single or double quotation marks.

.lib

Within a SPICE or included file, specifies a library file or section to be included. Within a library file, indicates the beginning of a library section.

T-Spice accepts two different library file formats. The **.lib** command is used to access library files of both formats. It is also used to delimit library sections within library files if the first library format is used. Specific model and subcircuit definitions are read in only if needed.

Syntax

Library File Format I:

The first T-Spice library file format is a sequence of library sections. Each library section begins with a **.lib** command and ends with a **.endl** command. The **.lib** command assigns a name to each section. Within each library section any sequence of SPICE circuit elements or commands may occur.

When **.lib** is used with both a file name and section name, it is equivalent to **.include** except that only the part of the file within the specified library section is included.

```
.lib filename [section]
```

filename	The file to be included. It must exist in the current directory or in the T-Spice search path. If the referenced filename or path contains a space, enclose the entire path in single or double quotation marks.
section	If specified, designates a section of the library file to be searched.

Examples

A file **test.lib** might contain:

```
.lib sub1
.subckt s1 a b
r1 a b 1k
.ends
.endl

.lib sub2
.subckt s2 a b
r1 a b 2k
.ends
.endl

.lib sub3
.subckt s3 a b
r1 a b 3k
.ends
.endl
```

The command:

```
.lib test.lib sub2
```

would cause T-Spice to include the library section **sub2** and therefore the definition for subcircuit **s2**.

Library file format II:

The second T-Spice library file format consists simply of a sequence of **.model** commands and **.subckt** definition blocks. A library file of this second format may be included in a simulation by using the **.lib** command in a main input file or included file. T-Spice will search the specified file for device model and subcircuit definitions if they are not found in the main input file or included files, and read in only those that are needed.

```
.lib filename
```

filename The library file to be searched for **.model** (page 85), **.param** (page 102), and **.subckt** (page 132) definitions. The file must exist in the current directory or in the T-Spice search path. If the referenced filename or path contains a space, enclose the entire path in single or double quotation marks.

Examples

A file **test2.lib** might contain:

```
.subckt s1 a b
r1 a b 1k
.ends

.subckt s2 a b
r1 a b 2k
.ends

.subckt s3 a b
r1 a b 3k
.ends
```

Suppose the main input file contains:

```
.lib test2.lib
x1 1 0 s3
```

The **.lib** command would cause the file **test2.lib** to be searched for model and subcircuit definitions. The instance **x1** references subcircuit **s3**, causing T-Spice to read and include the subcircuit definition for **s3** in **test2.lib**. Assuming that subcircuits **s1** and **s2** are not referenced elsewhere, their definitions in **test2.lib** would not be read in.

.load

Input the contents of the specified file. The file presumably was created using the **.save** (page 125) command, and contains either **.ic** (page 70) or **.nodeset** (page 89) commands for restoring the bias point of the circuit. The **.save** and **.load** commands can be used in combination to reduce simulation time by performing a compute-intensive operating point calculation once, saving the bias information, and then using the **.load** command in subsequent simulations to initialize the circuit to that state.

Syntax

```
.load [file=filename]
```

filename	Name of the file to be read. If the **file** parameter is not entered, then the filename is derived from the simulation input filename, with a **.ic** file extension.

Examples

```
.load file=baseline.ic
```

.macro /.eom

.macro is synonymous with **.subckt,** and .**eom** is synonymous with .**ends**.

The **.macro ... .eom** naming convention for defining subcircuits is provided for compatibiliity with other simulators which use this syntax rather than **.subckt ... .ends**.

Note: Prior to version 10.0 of T-Spice, the **.macro** command was used for creating table-based devices.
*This usage of the **.macro** command is no longer supported.*
Old circuit files which contain **.macro** table-based device definitions can be converted to the new syntax by replacing **.macro** statements with **x*name*** statements.

```
.macro mname tablename node1 node2 node3 ... noden
```

becomes:

```
xmname node1 node2 node3 ... noden tablename
```

.measure

Used to compute and print electrical specifications of a circuit, such as delay between signals, rise and fall times, and minimum and maximum values of a signal. Also used for optimization in conjunction with the following commands:

- **.ac** (page 51)
- **.dc** (page 60)
- **.connect** (page 57)
- **.step** (page 128)
- **.tran** (page 137)

For parameter sweeps, T-Spice generates a separate output section plotting measurement results versus swept parameter values. A data set can be used with the error measurement syntax of **.measure** to compute differences between simulation output curves and corresponding external data. In this case, ***out1*** or ***out2*** in the error function measurement syntax may refer to a column name of a **.data** statement.

A **.measure** command within a **.subckt** block is replicated for each instance of the subcircuit.

For optimization, you can use **.optgoal** instead of **.measure** with the **goal** and **minval** parameters to set the minimum value for the denominator in the error expression and the scalar value that specifies the relative importance of two or more measurements.

Note: When the **.options** (page 95) **autostop** field is set to 1, T-Spice automatically terminates any transient analysis when all **.measure** results have been found. The **autostop** option does not affect preview transient analyses.

Syntax

The **.measure** command syntax has several formats, each of which is described below. Each **.measure** command should be used in conjunction with DC transfer, AC, data, step, or transient analysis.

The general syntax of the **.measure** command is:

```
.measure {dc|ac|tran} result list [goal=goal]
   + [minval=minval] [weight=weight] [off]
```

dc | ac | tran — Denotes the analysis type (**dc**, **ac**, or **tran**) for which the measurement is to be done. For **dc** analysis, the independent variable is a swept parameter. For **ac** analysis, the independent variable is frequency. For transient analysis (**tran**), the independent variable is time.

result — Name of the measurement result. This name is used to identify the measurement result in the output file, and it can also be used in subsequent equation evaluation measurements.

list List of keywords and parameters whose syntax depends on the type of measurement to be made. Possible measurement types include:

- Trigger/Target Measurements on page 79
- Signal Statistics Measurements on page 80
- Find-When and Derivative Measurements on page 81
- Expression Evaluation Measurements on page 83
- Error Function Measurements on page 83

goal Desired value of the measurement, for use in optimization. In optimization, T-Spice attempts to minimize the relative error, (***goal***-***result***)/***goal***. For error measurement optimizations, T-Spice minimizes simply (***goal***-***result***), because in that case ***result*** is itself a relative value, and ***goal*** is usually zero. Default value: zero.

minval Minimum value for the denominator in the error expression above. Default: 1.0e-12.

weight Scalar value that is multiplied by the relative error. This value is used to specify the relative importance of two or more measurements in optimization. Default: 1.

off Optional keyword that prevents T-Spice from printing output from this **.measure** command.

Note: The **.measure** keyword can be abbreviated to **.meas**.

Trigger/Target Measurements

The trigger/target format of the **.measure** command is used to make independent variable (time, frequency, or swept parameter) difference measurements. The ***trigger*** and ***target*** specifications determine the beginning and end, respectively, of the measurement. The value of the measurement is the difference in the independent variable value between the trigger and the target. Common examples of trigger/target measurements include delay time, rise time, fall time, and bandwidth measurements.

The syntax of the ***parameters*** field for trigger/target measurements is:

```
trig outvar val=val [td=td] {cross=cross | rise=rise |
   fall=fall}
+      targ outvar val=val [td=td] {cross=cross | rise=rise | fall=fall}
```

or

```
trig at=at_value targ outvar val=val [td=td] {cross=cross | rise=rise |
   fall=fall}
```

at_value Specifies an explicit independent variable value for the trigger.

outvar Specifies an output signal on which the trigger or target measurement is performed. This can be any output plot item that is legal on a **.print** command for the appropriate analysis type. For **.measure ac** commands, **.print noise** plot items are allowed. ***outvar*** may be an output expression enclosed in single quotes.

val Specifies the value of ***outvar*** at which the trigger or target counter for ***cross***, ***rise***, or ***fall*** is incremented.

td Specifies a time delay before the measurement is enabled and crossings, rises, and falls are counted. Default: 0.

cross Indicates which occurrence of the trigger or target crossing is to be used for the measurement. A crossing occurs when the trigger or target ***outvar*** takes on the value *val*. The special syntax **cross=last** indicates that the last crossing is to be used.

rise Indicates which occurrence of the trigger or target rise crossing is to be used for the measurement. A rise crossing occurs when the trigger or target ***outvar*** takes on the value ***val*** while increasing. The special syntax **rise=last** indicates that the last rise crossing is to be used.

fall Indicates which occurrence of the trigger or target fall crossing is to be used for the measurement. A fall crossing occurs when the trigger or target ***outvar*** takes on the value ***val*** while decreasing. The special syntax **fall=last** indicates that the last fall crossing is to be used.

Note: For a particular trigger or target, only one of ***cross***, ***rise***, or ***fall*** may be specified.

Trigger/target measurement output reports the independent variable value difference between the trigger and the target, as well as the independent variable values at the trigger and target. The measurement result may be negative if the target independent variable value is less than the trigger independent variable value.

For syntax examples, see Trigger/Target Example on page 84.

Signal Statistics Measurements

Signal statistics measurements are used to perform data reduction operations on signals. T-Spice can compute average, RMS (root-mean-square), minimum, maximum, and peak-to-peak values for a signal. In addition, T-Spice can report the independent variable (argument) value at the minimum or maximum of a signal.

The syntax of the ***parameters*** field for signal statistics measurements is:

```
type outvar [from=from] [to=to] [output=output]
```

type Specifies the type of measurement, and is one of the following:

- **amax** computes the independent variable value at the point where the signal's maximum value is reached.

- **amin** computes the independent variable value at the point where the signal’s minimum value is reached. If the minimum value occurs more than once, **amin** reports the first instance.
- **avg** computes the average value of the signal, defined as the integral of the signal divided by the length of the measurement interval.
- **integral** computes the integral of the signal over the measurement interval. Can be abbreviated to **integ**.
- **max** computes the maximum value of the signal.
- **min** computes the (global) minimum value of the signal. The computation is independent of signal derivative (*i.e*., **min** may be a signal endpoint).
- **pp** reports the difference between the maximum and minimum values of the signal (peak-to-peak measurement).
- **rms** computes the root-mean-square value of the signal, defined as the square root of the integral of the signal squared, divided by the length of the measurement interval.

outvar Specifies the output signal on which the measurement is to be performed. It can be any output plot item that is legal on a **.print** command for the appropriate analysis type. For **.measure ac** commands, **.print noise** plot items are allowed. ***outvar*** may be an output expression enclosed in single quotes.

from Specifies the value of the independent variable (time, frequency, or sweep parameter) at the beginning of the measurement. The default is the beginning of the analysis.

to Specifies the value of the independent variable (time, frequency, or sweep parameter) at the end of the measurement. The default is the end of the analysis.

output For **max** and **min** type measurements, specifies the output signal to be evaluated at the point at which the maximum or minimum is reached.

For syntax examples, see Signal Statistics Example on page 84.

Find-When and Derivative Measurements

Find-when measurements are used to measure values of dependent or independent output variables when some specific event occurs. The event specification is similar to the trigger specification in trigger/target measurements: the event occurs when a signal crosses a value, or a signal crosses another signal, or when a certain independent variable “at” value is reached. If a “find” signal is specified, the measurement output is the value of the find signal at the event. If no “find” signal is given, then the measurement result is the independent variable value at the event.

Find-when measurements can also be used to compute derivatives of functions. If the **find** keyword is replaced with **derivative**, the derivative of ***outvar1*** is computed as the measurement result.

The syntax of the ***list*** field for find-when measurements is:

```
[find outvar1 | derivative outvar1] when outvar2=val [td=td] {cross=cross |
   rise=rise | fall=fall}
```

or

```
[find outvar1 | derivative outvar1] when outvar2=outvar3 [td=td]
   {cross=cross | rise=rise | fall=fall}
```

or

```
{find outvar1 | derivative outvar1} at=at_value
```

Note: For a particular trigger or target, only one of ***cross***, ***rise***, or ***fall*** may be specified.

at_value Specifies an explicit independent variable value for the event.

outvar1 Specifies the output signal to be evaluated as the measurement result at the event of interest. This can be any output plot item that is legal on a **.print** command for the appropriate analysis type. For **.measure ac** commands, **.print noise** plot items are allowed. ***outvar1*** may be an output expression enclosed in single quotes.

outvar2 Specifies an output signal to be evaluated for locating the measurement event. This can be any output plot item that is legal on a **.print** command for the appropriate analysis type. For **.measure ac** commands, **.print noise** plot items are allowed. ***outvar2*** may be an output expression enclosed in single quotes.

val Specifies the value of ***outvar2*** at which the event counter for crossings, rises, or falls is incremented.

outvar3 Specifies a second output signal to be evaluated for locating the measurement event. This can be any output plot item that is legal on a **.print** command for the appropriate analysis type. For **.measure ac** commands, **.print noise** plot items are allowed. ***outvar3*** may be an output expression enclosed in single quotes.

td Specifies a time delay before the measurement is enabled and crossings, rises, and falls are counted. Default: 0.

cross Indicates which occurrence of the event crossing is to be used for the measurement. A crossing occurs when ***outvar2*** takes on the value val or the value of ***outvar3***. The special syntax **cross=last** indicates that the last crossing is to be used.

rise Indicates which occurrence of the trigger or target rise crossing is to be used for the measurement. A rise crossing occurs when ***outvar2*** takes on the value val or the value of ***outvar3*** while increasing. The special syntax **rise=last** indicates that the last rise crossing is to be used.

fall Indicates which occurrence of the trigger or target fall crossing is to be used for the measurement. A fall crossing occurs when ***outvar2*** takes on the value ***val*** or the value of ***outvar3*** while decreasing. The special syntax **fall=last** indicates that the last fall crossing is to be used.

For syntax examples, see Find-When and Derivative Example on page 84.

Expression Evaluation Measurements

T-Spice can compute expressions that are functions of previous **.measure** command results. The syntax of the ***parameters*** field for expression evaluation measurements is:

param=*'expression'*

where ***expression*** is an algebraic expression involving **.param** parameter values, subcircuit parameter values, and previous **.measure** result names. The expression may not contain plot items such as node voltages or branch currents. The expression may contain the same operators and functions as used in **.print** output expressions.

Error Function Measurements

The error function measurement reports a relative difference between two output variables. Four methods for calculating this error are available.

The syntax of the ***parameters*** field for error function measurements is:

errtype out1 out2 [**from=***from*] [**to=***to*]

errtype Specifies the method of computing the total error. Must be one of **err**, **err1**, **err2**, **err3**.

out1, out2 Specifies the output signals to be compared. These can be any output plot items that are legal on a **.print** command for the appropriate analysis type. For **.measure ac** commands, **.print noise** plot items are allowed. ***out1*** and ***out2*** may be output expressions enclosed in single quotes.

from Specifies the value of the independent variable (time, frequency, or sweep parameter) at the beginning of the measurement. The default is the beginning of the analysis.

to Specifies the value of the independent variable (time, frequency, or sweep parameter) at the end of the measurement. The default is the end of the analysis.

The error computation depends on the ***errtype*** and is as follows:

err, err1 The error is the RMS value of the relative difference between the two signals, normalized to the length of the measurement interval.

err2 The error is the integral over the measurement interval of the absolute value of the relative difference of the two signals.

err3 The error is the integral over the measurement interval of the relative difference of the logs of the two signals.

The relative difference of the two signals is defined as:

(*out1*-*out2*)/max(*minval*, |*out1*|+|*out2*|)

The relative difference of their logs is defined as the absolute value of:

```
log(|out1|/max(minval, |out2|))/log(max(minval, |out1|+|out2|))
```

For syntax examples, see Error Function Example on page 84.

Examples

Trigger/Target Example

```
.measure tran delaytime trig v(1) val=2.5 fall=3
+      targ v(2) val=2.5 rise=3
```

measures the time delay from the third falling edge of signal **v(1)** to the third rising edge of signal **v(2)**. The measurement begins when **v(1)** falls through 2.5V for the third time, and ends when **v(2)** rises through 2.5V for the third time.

```
.measure tran risetime trig v(1) val=0.5 rise=1
+      targ v(1) val=4.5 rise=1
```

measures the first rise time of the voltage at node 1.

Signal Statistics Example

```
.measure ac maxgain max vm(out)
```

measures the maximum value of **vm(out)** over the frequency range covered during an AC analysis.

```
.measure ac resfreq amax vm(out)
```

measures the frequency at which the maximum value of **vm(out)** is achieved.

```
.measure ac phase_at_resonance max vm(out) output=vp(out)
```

measures the phase **vp(out)** at the frequency where **vm(out)** is at its maximum.

Find-When and Derivative Example

```
.measure tran v1 find v(1) when v(2)=2 cross=1
```

measures the voltage at node **1** when the voltage at node **2** crossed 2V for the first time.

```
.measure ac f1 when vm(out)=1
```

measures the frequency at which the voltage gain at node **out** is 1.

```
.measure tran d1 derivative v(1) at=100ns
```

measures the derivative of **v(1)** with respect to time at 100ns.

Error Function Example

```
.measure tran v1v2 err1 v(1) v(2)
```

measures the difference between the signals **v(1)** and **v(2)**.

.model

Specifies device model parameters to be used by one or more devices.

Syntax

```
.model modelname type [level=L][parameter=value [parameter=value [...]]]
   [ako: akomodel]
```

modelname	Model name.
type	Device type (see below).
L	Model level, required for device models with multiple levels.
parameter	The ***parameter*** list is predefined for each standard device model. (See the chapter Device Models on page 314.)

type is one of the following:

c	Capacitor
cpl	Coupled transmission line.
csw	Current-controlled switch element.
d	P-N diode.
npn	NPN-type BJT.
pnp	PNP-type BJT.
njf	N-type JFET.
pjf	P-type JFET.
nmf	N-type MESFET.
opt	Controls the optimization algorithm. For additional information, see Optimization Algorithm Parameters, below.
pmf	P-type MESFET.
nmos	N-type MOSFET.
pmos	P-type MOSFET.
r	Two-terminal resistor (or three-terminal resistor if a capacitance is specified).
sw	Voltage-controlled switch element.

external

External model. The parameter list of a **.model** command of type **external** must specify the name of the file that contains the external model. These special model parameters vary according to platform and take the following form:

- **winfile=*file*** (Windows)
- **solfile=*file*** (Solaris 2.*x*)
- **sunfile=*file*** (SunOS 4.*x*)
- **sgifile=*file*** (SGI IRIX)
- **hpfile=*file*** (HPUX)

If the external model filename has a **.c** extension, T-Spice will interpret the model file. If it has any other extension, T-Spice will treat it as a compiled DLL or shared library.

External model parameters can be numbers or strings. Strings and filenames must be enclosed in double quotes (**" "**).

Optimization Algorithm Parameters

The T-Spice optimization algorithm is controlled using the **.model** command with a model type of **opt**. The syntax is:

```
.model name opt [parameter=value [parameter=value […]]] [ako: akomodel]
```

modelname

Model name matched with the model name specified in an analysis command

parameter

Model parameter name

value

Value assigned to the model parameter

akomodel

Another **opt** model statement. If **akomodel** is specified, this model is considered to be “a kind of” the model defined with name **akomodel**; that is, all model parameters defined in the **akomodel .model** statement are included in this model unless overridden in this model.

Valid alternatives for ***value*** are listed below:

Parameter	*Description*	*Default*
cendif	Gradient value below which more accurate derivative computation methods are used to compute the gradient.	1e-9
close	Estimate of how close the optimization parameters’ initial value estimates are to the solution. **close** is a multiplier for computing new parameter estimates. Larger values result in larger steps toward the solution.	0.001

cut	Modifies **close** from one iteration to the next. If an iteration was unsuccessful, **close** is divided by **cut**; if an iteration is successful, **close** is multiplied by **cut** squared.	2
difsiz	Determines the increment in a parameter value used to compute numerical derivatives. The increment used is **difsiz*max**(***value***, **parmin**), where ***value*** is the parameter value. If **delta** is specified on the **.param** command, then **delta** is used as the increment.	1e-3
grad	Convergence tolerance for the gradient. If the gradient is less than ***grad***, then convergence may have been achieved, depending on the outcome of the additional **relin** and **relout** tests.	1e-6
itropt	Maximum number of optimization loop iterations.	20
level	Optimization method used. **Level=1** selects a modified Levenberg-Marquardt algorithm.	1
max	Upper limit for **close**	600,000
parmin	Used in increment selection for derivative computation; see **difsiz** parameter, above	0.1
relin	Relative input parameter variation for convergence. If all optimization parameters (defined with **.param**) vary by less than this amount (0.1% by default) from one iteration to the next, convergence is declared	0.001
relout	Relative output parameter variation for convergence. If the total error defined by measurement results varies by less than this amount from one iteration to the next, convergence is declared.	0.001

Note: When no model parameters are specified, all models take on their default values.

Examples

```
.model nmos nmos
+ Level=2            Ld=0.0u             Tox=225.00E-10
+ Nsub=1.066EV+16    Vto=0.622490        Kp=6.326640E-05
+ Gamma=.639243      Phi=0.31            Uo=1215.74
+ Uexp=4.612355E-2   Ucrit=1746677       Delta=0.0
+ Vmax=177269        Xj=.9u              Lambda=0.0
+ Nfs=4.55168E+12    Neff=4.68830        Nss=3.00E+10
+ Tpg=1.000          Rsh=60              Cgso=2.89E-10
+ Cgdo=2.89E-10      Cj=3.27E-04         Mj=1.067
+ Cjsw=1.74E-10      Mjsw=0.195
```

Specifies the parameters for an *n*-type MOSFET model called **nmos**.

```
.model resmodel external winfile="res.dll" solfile="res.sl" a=20 b="name"
```

Defines a model called **resmodel** that can be referred to by an instance (**x**) statement. In Windows, **res.dll** is loaded as a DLL; under Solaris **res.sl** is loaded as a shared library. Under all other platforms, an error message is issued because no external model name is specified. The numeric parameter **a=20** and the string parameter **b="name"** are passed to the external model as model parameters.

.nodeset

Sets an “initial guess” for the iterative DC operating point calculation.

- DC operating points are calculated by the **.ac** (page 51), **.dc** (page 60), **.op** (page 91), **.tf** (page 136), and **.tran** (page 137) commands.
- DC operating points are calculated by iteration, and **.nodeset** sets starting points for iteration. Convergence properties can be very sensitive to the initial guess; **.nodeset** can be used (1) to enable convergence for difficult-to-converge circuits and (2) to cause T-Spice to converge to one particular solution if more than one solution exists.
- After **numndset** iterations, or when the convergence criteria have been met, the specified nodes are allowed to float.
- Nodes and devices within subcircuits can be accessed with hierarchical notation in the form **x*instance*.x*instance*.*node***.
- To set node voltages for the duration of the operating point calculation, use the **.ic** (page 70) command.
- **.nodeset** commands inside subcircuit definition blocks are replicated for each subcircuit instance.

Syntax

```
.nodeset node=X [[,] node=X ...]
.nodeset v(node)=X [[,] v(node)=X ...]
```

node	Node to be initialized.
X	Node voltage relative to ground. (Unit: volts.)

Examples

```
.nodeset n1=5V n2=2
.nodeset v(n1)=5V v(n2)=2
```

The two examples are identical except for syntax.

.noise

Computes the effect of circuit noise on output voltage in conjunction with AC analysis.

- If the **.ac** (page 51) command is missing from the input file, the **.noise** command is ignored.
- Noise analysis is performed at the same frequencies as specified by the **.ac** (page 51) command.
- Noise models take the form of frequency-dependent mean-square currents (since the underlying phenomena are “random”) generated by adding a current source to the circuit for each modeled noise source.
- Noise sources at different points in the circuit are uncorrelated.
- Noise models are available for resistors and semiconductor devices (diodes, BJTs, JFETs, MESFETs, and MOSFETs). Semiconductor device models may contain noise model parameters which affect the size of noise sources.
- External model devices may also contain noise sources.
- Noise analysis results can be reported with the **.print** (page 108) **noise** command.

Syntax

```
.noise v(node1 [[,] node2]) source interval
```

node1 Output node.

node2 Reference node. (Default: ground.)

source Input voltage or current source, at which noise can be considered to be concentrated for the purposes of estimating the equivalent noise spectral density.

interval Report interval. A noise report will be printed to the simulation log which lists every device and the noise contribution and noise components. This report will be printed for the first frequency and each *interval* frequency. (Default: 0)

.op

Performs a DC operating point calculation and outputs all node voltages and voltage source currents.

- DC operating points are calculated on the assumption that there are no charge effects in the system: capacitors are open and inductors are shorted.
- The **.ic** (page 70) command can be used to impose initial conditions on nodes. Initial conditions are represented by voltage sources present for the duration of the DC operating point calculation, and removed for transient simulations.
- The **.nodeset** (page 89) command can be used to set initial guesses for the iterative solution process for DC operating points.
- The results of the **.op** command are written automatically to the specified output file. Other results can be reported with the **.print** (page 108) **dc** command.
- Small-signal transfer function data can be reported with the **.tf** (page 136) command.
- Small-signal parameters are automatically reported to the output file. You can specify a separate output file for small-signal data with the **.acmodel** (page 54) command. The command **.acmodel {}** disables small-signal parameter reporting. Reporting is also disabled if the simulation model has more than 1000 nodes.

Syntax

```
.op [noprint]
```

noprint Turns off automatic **.op** output.

.optgoal

Sets optimization goals. Allows the same measurement to be used in different optimization runs with different goals and weight values.

Note that during an optimization run, all **.optgoal** commands with the same ***optname*** are used as optimization results, in addition to any measurements specified in the results list on the sweep optimize syntax of a **.step** or one of the analysis commands. (See the following example.)

If the **.measure** command does not exist in the netlist, an error message will be returned.

The formula for optimization functions is:

$$\sum_k \left(W_k \cdot \frac{(G_k - M_k)}{max(minval, |G_k|)} \right)^2 \qquad (0.6)$$

where *Wk* is the weight, *Gk* is the goal, and *Mk* is the measurement value.

Syntax

```
.optgoal optname measname=goal [minval=minval] [weight=weight]
```

optname Name of the optimization run.

measname A **.measure** result to be used as the goal.

goal Optimization goal value.

minval The minimum denominator value for the optimization error computation. Defaults to the value in the **.measure** command identified by ***measname***, or if not specified there, to1.0e-12.

weight A relative importance of the optimization goal with respect to other goals for the same optimization run. Defaults to the value in the **.measure** command identified by ***measname***, or if not specified there, to1.

Examples

```
.ac dec 10 1 100k sweep optimize=opt1 results=gain model=optmod
.measure ac gain max vdb(out) goal=30
.measure ac bandwidth when vdb(out)=10
.optgoal opt1 bandwidth=5k
```

results in both the gain and bandwidth measurements contributing to the overall optimization goal.

.optimize

Invokes an optimization run using parameters and goals specified using **.paramlimits** and **.optgoal** commands with the same ***optname***.

.ac, **.dc**, **.step** and **.tran** use the parameter **analysis** to identify analysis commands from the **.optimize** command. However, it is possible to avoid assigning an analysis name. If the analysis name on the **.optimize** command is the name of an analysis type ("ac", "tran", "dc", or "step",) and no analysis of that name exists, T-Spice will perform the optimization on the first analysis of that type. (See Examples on page 93.)

Syntax

```
.optimize optname model=modelname analysis=analysisname
```

optname

Name of the optimization run.

modelname

Refers to a **.model** command of type ***opt*** which specifies optimization algorithm parameters such as iteration count limits and convergence tolerances.

analysisname

Identifies a **.step**, **.ac**, **.dc**, or **.tran** command of the same name that will be performed to evaluate the measurements for the optimization.

Examples

```
.ac dec 10 1 100k analysisname=ac1
.optimize opt1 model=optmod analysisname=ac1
```

invokes an optimization around an AC analysis. This AC analysis optimization syntax is equivalent to:

```
.ac dec 10 1 100k
.optimize opt1 model=optmod analysisname=ac
.param r=1k c=1u
r1 1 0 'r'
c1 1 0 'c'
.options autostop reltol=1e-6
.ic v(1)=1
.tran 0.1m 100m
.print tran v(1)
.measure tran decaytime when v(1)=0.5
```

Optimization commands:

```
.optimize opt1 model=optmod analysisname=tran
.model optmod opt level=1 itropt=40
.optgoal opt1 decaytime=300u
.paramlimits opt1 r minval=10    maxval=100k
.paramlimits opt1 c minval=0.01u maxval=100u
.end
```

This is a transient analysis of a simple RC circuit. The **.measure** command measures the amount of time it takes for the voltage at node 1 to decay to half its initial value. Theoretically, this decay time is equal to **R*C*ln(2)**. The **.optimize** command invokes an optimization run called **opt1**. The **.optgoal** command sets the optimization goal (**300u**sec) for the decay time, and the **.paramlimits** commands specify ranges for the **R** and **C** parameters.

During the optimization run, **R** and **C** can be varied within their ranges in order to achieve the decay time goal of 300 microseconds. The output for this simulation is as follows:

Optimized parameter values:

```
   r =   6.5783e+02
   c =   6.5775e-07

* END NON-GRAPHICAL DATA

*WEDIT: .tran   0.0001      0.1
TRANSIENT ANALYSIS - OPTIMIZE=opt1
Time<s>            v(1)<V>
    0.0000e+00   1.0000e+00
    1.9207e-08   9.9996e-01
    1.8027e-06   9.9584e-01
    3.4447e-06   9.9207e-01
    1.7692e-05   9.5994e-01
    2.6271e-05   9.4109e-01
    3.4134e-05   9.2415e-01
    4.2453e-05   9.0655e-01
    5.3935e-05   8.8281e-01
    7.0558e-05   8.4953e-01
    8.9536e-05   8.1308e-01
    1.2194e-04   7.5440e-01
    1.5149e-04   7.0461e-01
    1.8663e-04   6.4965e-01
    2.2571e-04   5.9354e-01
    2.6455e-04   5.4257e-01
    3.0212e-04   4.9745e-01

* BEGIN NON-GRAPHICAL DATA

MEASUREMENT RESULTS - OPTIMIZE=opt1

decaytime = 3.0000e-04

* END NON-GRAPHICAL DATA
```

Note that the optimized **R** and **C** values achieve the optimization goal of **300u**. This is consistent with theoretical prediction: for the optimized **R** and **C** values of 657.83 Ohms and 657.75 nanofarads, the theoretically predicted decay time is 299.92 microseconds.

.options

Sets global simulation options.

Syntax

```
.options field=X [field=X ...]
```

Fields that toggle actions on or off can be specified as **true/false**, **t/f**, **1/0,** or **yes/no**. Specifying a **true/false** field without a value automatically sets the field to **true**.

Option fields are described in detail in the following chapter, Simulation Options on page 194. The following tables list a summary of simulation options; click on the option name for a full description.

Accuracy and Convergence Options

Field	*Description*	*Default*
absi | abstol (page 196)	Maximum allowed RMS of residual branch currents when **kcltest** = **true**.	1×10^{-10} A
absv | vntol (page 197)	Maximum absolute node voltage change allowed between iterations when **kvltest** = **true**.	1×10^{-6} V
accurate (page 198)	Triggers changes to other option settings to maximize simulation accuracy.	**false**
bypass (page 199)	Controls the diode and transistor bypass algorithm	**true**
bytol (page 200)	Sets the relative tolerance for the bypass algorithm terminal voltage values	0.0
cshunt (page 201)	Capacitance added from each node to ground.	0.0 F
dchomotopy (page 202)	Algorithm used to correct DC operating point non-convergences. Can be set to **none**, **source**, **gmin**, **pseudo**, or **all**.	**all**
dcmethod (page 204)	Default method for solving a DC operating point problem. Can be set to **standard**, **source**, **gmin**, or **pseudo**.	**standard**
dcstep (page 205)	Controls the conductance added across the terminals of each capacitor during DC operating point computation. (*g*=*c*/***dcstep***, where *c* is the device capacitance.)	0.0
extraiter[ations] | newtol (page 206)	Number of additional iterative steps to calculate after convergence criteria have been met.	0
fast (page 207)	Triggers changes to other options settings to maximum simulation speed.	**false**
gmin (page 208)	Conductance added in parallel with all *pn* junctions during transient analysis.	$1 \times 10^{-12}\,\Omega^{-1}$
gmindc (page 209)	Conductance added in parallel with all *pn* junctions during DC operating point analysis.	$1 \times 10^{-12}\,\Omega^{-1}$

Field	*Description*	*Default*
gramp (page 210)	Specifies the range over which the gmindc variable is swept during g_{min} stepping. (***gmindc*** $\le g_{min} \le$ ***gmindc*** × $10^{\textbf{\textit{gramp}}}$)	4
gshunt (page 211)	Conductance added from every node to ground.	0.0 Ω^{-1}
kcltest (page 212)	Enables the current tolerance test for convergence.	**true**
kvltest (page 213)	Enables the voltage tolerance test for convergence in transient analysis (always true for DC analysis).	**false**
maxdcfailures (page 214)	Maximum number of non-convergence failures allowed in a DC sweep simulation.	4
mindcratio (page 215)	Minimum fractional step size allowed in source ramping for DC sweep analysis.	1×10^{-4}
minsrcstep (page 216)	Minimum fractional step size for source stepping in DC operating point analysis.	1×10^{-8}
numnd | itl1 (page 217)	Newton iteration limit for DC operating point computation.	250
numndset (page 218)	The maximum number of Newton iterations during which the **.nodeset** nodes will be held at their user-specified voltage values.	**numnd** / 10
numns | itl6 (page 219)	Newton iteration limit for source stepping attempts in DC operating point analysis.	50
numnx | itl2 (page 220)	Newton iteration limit for DC sweep computation.	100
numnxramp (page 221)	Newton iteration limit for DC sweep computation during source ramping.	50
precise (page 222)	Triggers changes to other options settings to maximize simulation precision.	**false**
reli | reltol (page 224)	Maximum relative change in RMS branch current allowed between iterations when **kcltest** = **true**.	5×10^{-4}
relv (page 225)	Maximum relative change in node voltage allowed between iterations when **kvltest** = **true**.	1×10^{-3}
tolmult (page 226)	Multiplicative scaling factor for **absi**, **absv**, **reli**, and **relv**.	1.0

Timestep and Integration Options

Field	*Description*	*Default*
absdv | absvar (page 228)	Threshold absolute voltage change between two consecutive time steps; used to calculate voltage variance when **lvltim**=1 | 3 | 4.	0.5 V
absq | chgtol (page 229)	Minimum capacitor charge or inductor flux used to compute Local Truncation Error for timestep control (**lvltim**=2).	1×10^{-4} C
ft (page 230)	Fraction by which the timestep is reduced if a transient analysis solution does not converge within ***numnt*** iterations.	0.4

Field	*Description*	*Default*
lvltim (page 231)	Algorithm used to control timestep sizes in transient analysis: 1 = Iteration count algorithm with voltage variance test 2 = Local Truncation Error algorithm 3 = Iteration count algorithm with voltage variance and timestep reversal 4 = voltage variance test for timestep prediction plus Local Truncation Error algorithm for timestep reversal (hybrid of lvltim 1 and 2)	1
maxord (page 234)	Maximum time integration order for Gear's BDF calculation.	2
method (page 235)	Method of numerical integration for estimating time derivatives of charge during transient analysis.	**trap**
mintimeratio | rmin (page 236)	Relative minimum timestep size for transient simulations.	1×10^{-9}
mu | xmu (page 237)	Coefficient for varying integration between the backward Euler formula and the trapezoidal formula. Used when **method** = **trap**.	0.5
numnt | itl4 | imax (page 238)	Newton iteration limit for transient analysis solutions.	10
numntreduce | itl3 (page 239)	Threshold number of Newton iterations for controlling decrease or increase of the next timestep.	3
poweruplen (page 240)	Length of the powerup ramp during powerup transient analysis.	0.1% total **.tran** time
reldv | relvar (page 241)	Maximum relative voltage change between time steps; used to calculate the voltage variance when **lvltim** = 1 | 3.	0.35
relq | relchgtol (page 242)	Maximum relative error in predicted charge; used to adjust timestep sizes in the LTE algorithm (**lvltim** = 2).	5×10^{-4}
rmax (page 243)	Maximum allowed timestep, given as a multiple of the timestep specified with **.tran**.	2
trextraiter[ations] | trnewtol (page 244)	Number of additional iterative steps to calculate at each timestep after convergence criteria have been met	0
trtol (page 245)	Corrective factor for estimation of the local truncation error (LTE) when **lvltim**=2.	10

Model Evaluation Options

Field	*Description*	*Default*
dcap (page 247)	Model selector for calculating depletion capacitances.	2
dccap (page 248)	Flag to compute device charge and capacitance values in DC analysis	**false**
defad (page 249)	Default MOSFET drain diode area.	0.0 m^2
defas (page 250)	Default MOSFET source diode area.	0.0 m^2

Field	*Description*	*Default*
defl (page 251)	Default MOSFET channel length.	1×10^{-4} m
defnrd (page 252)	Default number of diffusion squares for a MOSFET drain resistor.	0.0
defnrs (page 253)	Default number of diffusion squares for a MOSFET source resistor.	0.0
defpd (page 254)	Default MOSFET drain diode perimeter.	0.0 m
defps (page 255)	Default MOSFET source diode perimeter.	0.0 m
deftables (page 256)	Selects use of table-based evaluation of models instead of direct model evaluation. See also **modelmode**.	**false**
defw (page 257)	Default MOSFET channel width.	1×10^{-4} m
deriv (page 258)	Selects the method for computing *dq/dv* and *di/dv* derivatives.	0
minresistance | resmin (page 259)	Minimum (floor) resistance value for all resistors.	$1 \times 10^{-5}\,\Omega$
modelmode (page 260)	Model evaluation method (**direct**, **uniform**, or **cache**).	**direct**
moscap (page 261)	Enables automatic source/drain area/perimeter estimation for MOSFETs.	**false**
mosparasitics (page 262)	Controls explicit modeling of diodes during table-based model evaluation mode.	**true**
scale (page 263)	Scales the physical dimensions of capacitors, MESFETs, MOSFETs, and resistors.	1.0
scalm (page 264)	Default scaling factor for resistors and capacitors.	1.0
tnom (page 265)	Nominal temperature.	25 deg. C
wl (page 266)	Reverses MOSFET length and width specifications.	**false**

Linear Solver Options

Field	*Description*	*Default*
linearsolver (page 268)	Selects the linear equation solver — **best**, **sparse**, or **superlu**.	**best**
pivtol (page 269)	Minimum pivoting tolerance for real matrices.	1×10^{-14}
zpivtol (page 270)	Minimum pivoting tolerance for complex matrices.	1×10^{-6}

General Options

Field	*Description*	*Default*
autostop (page 272)	Terminates transient analysis after all **.measure** results have been found.	**false**
casesensitive (page 273)	Case sensitivity for names of models, subcircuits, library sections, parameters, and nodes.	**false**

Field	*Description*	*Default*
compatibility (page 274)	Specifies input syntax and option setting compatibility with other simulators - Berkeley SPICE, HSPICE, or PSPICE.	**HSPICE**
conncheck (page 275)	Enables connectivity checking.	**true**
parhier (page 276)	Establishes the scoping algorithm for selection of parameter values in a hierarchical design.	**local**
persist (page 278)	Instructs T-Spice to continue simulation when the specified levels of warnings or errors are generated.	1
search (page 279)	Search path for library and include files.	
spice (page 280)	Changes other option settings to be compatible with Berkeley SPICE.	**false**

Output Options

Field	*Description*	*Default*
acct (page 282)	Tracks and reports iteration counts and other accounting statistics.	**false**
acout (page 283)	Calculation method for AC magnitude/phase differences.	1
brief (page 284)	Minimizes the amount of diagnostics printout which is written to the simulation status window.	**false**
captab (page 285)	Lists capacitances for each node in the netlist.	**false**
csdf (page 286)	Generates output in CSDF mode.	**false**
dnout (page 287)	Selects noise spectral density units.	0
echo (page 288)	Causes T-Spice to print each line of input to the error log as it is read.	**false**
expert (page 289)	Produces a listing of node and device convergence residual information.	**false**
ingold (page 290)	Controls the format of numbers printed in the AC small-signal output and the device listings. (0=engineering format, 1=g format, 2=e format)	**0**
list (page 291)	Produces a listing of all circuit devices.	**false**
maxmsg (page 292)	Sets the maximum number of duplicate warning message printouts.	5
node (page 293)	Prints a node cross-reference table.	**false**
nomod (page 294)	Controls the printout of diode and transistor models. Set nomod to 1 (true) to disable printout.	1
numdgt (page 295)	Minimum number of decimal places included in each **.print** output value.	4
nutmeg (page 296)	Generates output compatible with the *Nutmeg* graphics program.	**false**
opts (page 297)	Prints the settings of all control options.	**false**
outputall (page 298)	Causes all listings of nodes, devices, or options to include items that are internal or normally hidden to the user.	**false**
pathnum (page 299)	Prefixes subcircuit node and element names with a number rather than the full subcircuit path name.	**false**
prtdel (page 300)	Fixed time delay between output points in transient analysis.	0.0 s
prtinterp (page 301)	Determines how solutions are calculated at time intervals set by **prtdel**.	0
statdelay (page 302)	Minimum delay (real time) between status display updates in the T-Spice GUI.	0.5 s
tabdelim (page 303)	Toggles tab-delimited output columns.	**false**
verbose (page 304)	Level of circuit and simulation detail printed to the Simulation Window.	1

Field	*Description*	*Default*
xref (page 305)	Generates a listing of various circuit cross-reference information: conditional statement tree, symbol definitions, subcircuit listings.	**false**

Probing Options

Field	*Description*	*Default*
binaryoutput (page 307)	Specifies the form of binary output created with **.probe**.	3
probei (page 308)	Includes device terminal current values in the output data from **.probe** and **.print** (when used without arguments).	**false**
probeq (page 309)	Includes device terminal charge values in the output data from **.probe** and **.print** (when used without arguments).	**false**
probev (page 310)	Includes node voltage values in the output data from **.probe** and **.print** (when used without arguments).	**false**
probefilename (page 311)	Filename for binary output produced by **.probe**.	*outputfile*.dat
probesdbfile (page 312)	S-Edit design file from which the SPICE netlist was exported.	
probetopmodule (page 313)	S-Edit design module from which the SPICE netlist was exported.	

.param

Defines and assigns values to parameters, or creates a user-defined function.

- Parameters can be used in expressions to replace numeric values.
- User-defined functions are very similar to the set of built-in algebraic functions. User-defined functions may take any number of arguments, and are defined using an algebraic expression which performs operations on the function arguments.
- The scope of defined parameters extends to files referenced by **.if ... / .elseif ... / .else / .endif** (page 71) commands.
- Parameters placed in subcircuit definition blocks are valid only within that subcircuit definition, and override global parameters of the same name.
- Parameters placed outside subcircuit definition blocks are valid globally.

Note: To view a listing of all parameters that are defined in the input files, use the command **.option xref**.

Syntax

```
param parameter={X |mc_distribution|opt_limits}[[,]
   {X |mc_distribution|opt_limits} ...]
```

parameter Parameter name.

X Any number or valid expression. Expressions must be enclosed by single quotes.

mc_distribution Defines probability distributions used in Monte Carlo iterations. For additional information, see Monte Carlo Parameters, below.

opt_limits Defines optimization parameters. For additional information, see Optimization Parameters on page 104.

User-Defined Functions

The syntax for defining a function is:

```
.param funcname( [arg1 [, arg2 [, arg3 [...] ] ] ] )='body'
```

funcname The name of the user function, which may not be the same as a built-in algebraic function, or the same as a parameter name.

arg1 arg2 ... Function arguments which will be passed into the function from the function reference.

body An algebraic expression which solves the functional equation.

Monte Carlo Parameters

For each Monte Carlo iteration, T-Spice will report the values of all expressions evaluated using probability distributions defined by the following syntax:

```
.param parameter=unif(nominal_val, rel_variation [, multiplier])
```

or

```
.param parameter=aunif(nominal_val, abs_variation [, multiplier])
```

or

```
.param parameter=gauss(nominal_val, rel_variation, sigma [, multiplier])
```

or

```
.param parameter=agauss(nominal_val, abs_variation, sigma [, multiplier])
```

or

```
.param parameter=limit(nominal_val, abs_variation)
```

parameter Name of the parameter to be varied in the Monte Carlo analysis.

unif Selects a uniform distribution with relative variation specification.

aunif Selects a uniform distribution with absolute variation specification

gauss Selects a Gaussian distribution with relative variation specification.

agauss Selects a Gaussian distribution with absolute variation specification.

limit Selects a random limit distribution function using absolute variation. The result is either ***nominal_val***-***abs_variation*** or ***nominal_val***+***abs_variation***, with 50% probability for each.

nominal_val Nominal value for the parameter.

abs_variation Largest deviation from ***nominal_val*** that can be obtained from a uniform or limit distribution, or the standard deviation multiplied by ***sigma*** for a Gaussian distribution.

rel_variation Relative variation specification. The corresponding absolute variation is ***rel_variation********nominal_val***.

sigma Sigma-level at which the absolute or relative variation is specified for a Gaussian distribution. For example, if ***sigma***=3, the standard deviation is ***abs_variation***/3.

multiplier Number of times the distribution function is evaluated. The largest deviation from the nominal value of all evaluations is the one that is used as the result. The resulting distribution is bimodal. (*Default*: 1.)

Note: Multiple parameters can be assigned on the same **.param** command. Probability distributions are reevaluated with every use of ***paramname*** in expressions.

Optimization Parameters

When **.param** is used to define optimization parameters, the ***parameter*** argument uses the following syntax:

```
.param parameter=optname(guess, min, max [, delta])
```

parameter Global parameter name.

name References a particular optimization run name.

guess Initial (nominal) value for the parameter.

min Minimum values the parameter can take on.

max Maximum values the parameter can take on.

delta Used for discrete optimization. The final parameter values must differ from the initial guess by an integer multiple of *delta*. This is useful for optimizing quantities that can only take on discrete values, such as transistor lengths and widths.

Note: Multiple optimization parameters can be assigned on the same **.param** command.

The parameter ***parameter*** is allowed to vary within its range when an optimization of the appropriate ***name*** is invoked on an analysis command. During such an optimization, the parameter is initially assigned its ***guess*** value, but is allowed to vary within its range (defined by ***min*** and ***max***) during subsequent optimization iterations.

For additional information on the optimization syntax for individual analysis commands, see **.ac** (page 51), **.dc** (page 60), or **.tran** (page 137).

Examples

```
.param pi='4*atan(1)' tf='1E-6*sin(pi/2))'
.tran 'tf*0.01' 'tf'
```

The **.param** command defines and assigns a value to parameter **tf**, which is subsequently used (enclosed by single quotes) in place of a numeric value in the **.tran** (page 137) command.

```
.param res=agauss (100, 10, 1)
```

Specifies that the resistance is chosen from a normal distribution of mean 100 and standard deviation 10.

```
.param w1=opt1 (10u, 2u, 20u, 0.25u)
```

Specifies that **w1** is to be varied in optimization run **opt1** within the limits 2 × 10^{-6} and 20 × 10^{-6}. The initial guess for **w1** in the optimization is 10 × 10^{-6}, and the final value will be a multiple of 0.25 × 10^{-6}.

```
.param safedivision(a,b)='if(abs(b)<1e-100, 1e100, a/b)'
.print tran impedance='safedivision(v(n1), i1(dev1))'
```

Creates a function named safedivision which divides one number by another without a *division by zero* error. This function is then used in a print expression.

.paramlimits

Sets optimization parameter ranges. Allows the same parameter to be varied in multiple optimizations with different optimization run names.

This command specifies that the parameter ***paramname*** (specified using **.param** elsewhere) is to be varied and optimized during an optimization run ***optname***. Multiple instances of **.paramlimits** may not exist in the netlist for the same optimization run and the same parameter name, but are allowed for the same ***optname*** but for different ***paramname*** values.

Note: T-Spice supports sequential optimization—multiple optimizations may be performed in series from one input file, and the optimization results used in subsequent optimizations.

Syntax

```
.paramlimits optname paramname [guessval=guess] minval=min maxval=max
   [delta=delta]
```

optname Name of the optimization run.

paramname References a particular parameter.

guess Initial (nominal) parameter value for the optimization. If not used, the initial value defaults to the value specified in the **.param** command.

min Minimum value of the range in which the parameter may vary.

max Maximum value of the range in which the parameter may vary.

delta If specified, the parameter can change only in integer multiples of this value. This is useful for optimizing quantities which can only take on discrete values, such as transistor lengths and widths.

Examples

```
.paramlimits opt1 r minval=10 maxval=100k
.paramlimits opt1 m1width minval=1u maxval=10u delta=0.25u
```

.power

Computes power dissipation in conjunction with transient analysis.

- If the **.tran** (page 137) command is missing from the input file, the **.power** command is ignored.
- The average power consumption, the instantaneous maximum power, and the instantaneous minimum power (in watts) and the times of maximum and minimum power consumption (in seconds) are reported at the end of the transient simulation.
- The instantaneous power $P(t)$ dissipated by a voltage source at time t is the current through the source multiplied by the voltage drop across the source. The average power P for a time interval ($t1$,$t2$) is computed by using the trapezoidal rule approximation to evaluate the integral

$$\bar{P}(t_1, t_2) = \frac{1}{t_1 - t_2} \int_{t_1}^{t_2} P(\tau)\, d\tau \qquad (0.7)$$

- Multiple **.power** commands can be used in a single simulation.
- Power results can also be reported with the **.print** (page 108) **tran** command.

Syntax

```
.power source [A [Z]]
```

source Voltage source whose power consumption is to be computed.

A Time at which power recording begins. (Unit: seconds. Default: simulation start time.)

Z Time at which power recording ends. (Unit: seconds. Default: simulation end time.)

Examples

```
.power vtest 3e-7
```

Computes the power dissipated by voltage source **vtest** between the given time (0.3 microsecond) and the end of the simulation. It might produce the following sample output:

```
Power Results
vtest from time 0 to 3e-007
Average power consumed -> 1.249948e-002 watts
Max power 2.490143e-002 at time 9.76e-006
Min power 2.282279e-030 at time 1e-005
```

.print

Reports simulation results.

- If an output filename is not specified, the output file specified in the Start Simulation dialog is used. (If that file cannot be opened, the results are written to the Simulation Window.)
- Multiple **.print** commands can be used to direct different types of output to separate files.
- Transient analysis results are printed in columns, with time values in the first column.
- AC and noise analysis results are printed in columns, with frequency values in the first column.
- DC transfer analysis results are printed in columns, with sweep values from the first source listed on the **.dc** (page 60) command in the first column.
- DC operating point analysis results are printed line by line, each argument to a line.
- Expressions can be printed by themselves, with or without reference to physical quantities at specific nodes.
- Nodes and devices within subcircuits can be accessed with hierarchical notation in the form **x*instance*.x*instance*.*node***.
- **.print** commands inside subcircuit definition blocks are replicated for each instance.
- If no arguments are given, all node voltages and source currents are printed. If neither mode or arguments are given, **.print** applies to all analysis types. If arguments are given, a mode must also be specified.
- Wildcards may be entered as part of the **.print** command node names, devices names, terminal names, or terminal numbers. Wildcards are expanded to match any available elements which match the name specification.
- *Device State* print statements are a means of obtaining very detailed information about devices and device internal states. Data such as the current, charge, capacitance, and voltage values can be listed, as well as certain model evaluation variable (threshold voltage, beta, etc.).

Syntax

```
.print [mode ] ["filename"] [arguments ]
```

mode Analysis mode (see below).

filename Output filename. Must be enclosed by double quotes.

arguments Information to be printed (see below). ***arguments*** may include valid expressions involving other arguments or global parameters.

mode is one of the following:

tran Print results from transient analysis.

dc Print results from DC transfer analysis and DC operating point analysis.

ac Print results from AC analysis.

noise Print results from noise analysis.

arguments take one or more of a number of values, depending on ***mode***. When no arguments are given, all node voltages and source currents are printed.

When an argument includes an expression, the expression must be enclosed by single quotes (' '). A string can also be used as a column heading in the output file, and the string can be followed by an optional unit specifier, enclosed by angle brackets (**< >**). The unit is then displayed on the W-Edit *y*-axis.

Some entries in the argument tables below involve the variable ***z***, to be replaced by a key letter or number representing a device terminal. The key letters and numbers corresponding to particular device terminals are as follows (alternatives are separated by slashes):

- *Diodes*: anode = **P/1**; cathode = **N/2**.
- *BJTs*: collector = **C/1**; base = **B/2**; emitter = **E/3**; substrate = **S/4**.
- *JFETs/MESFETs*: drain = **D/1**; gate = **G/2**; source = **S/3**.
- *MOSFETs*: drain = **D/1**; gate = **G/2**; source = **S/3**; bulk = **B/4**.

Wildcards provide an easy and compact method of printing a large number of node or device values which have related names. The T-Spice **.print** command supports several types of wildcards in the specification of the node name, device name, terminal number, and terminal name. The ’*’ character (asterisk) will be expanded to match any combination of alpha-numeric characters. The ’?’ character (question mark) will be expanded to match any single alpha-numeric character. And, ’[...]’ will be expanded to match any single character enclosed within the square brackets.

The *arguments* for transient, transfer, and DC analysis (**.print tran**, **.print dc**) are as follows.

n — Voltage at node ***n*** relative to ground.

i(*d*,*n*) — Current at node ***n*** of device ***d*** (inward current positive).

iz(*d*) — Current at terminal ***z*** of device ***d*** (inward current positive).

p(*d*) — Power consumed by voltage source ***d***. This result can also be reported with the **.power** command.

q(*d*,*n*) — Charge at node ***n*** of device ***d***.

qz(*d*) — Charge at terminal ***z*** of device ***d***.

v(*n1*[[,]***n2*])** — Voltage at node ***n1*** relative to node ***n2***. (Default reference node: ground.)

’time()’ — Simulation time. Must be or be part of an expression enclosed by single quotes. (Transient analysis only.)

The *arguments* for AC analysis mode (**.print ac**) are as follows.

idb(*d*,*n*) — Current magnitude at node ***n*** of device ***d***. (Unit: decibels.)

idb*z*(*d*) — Current magnitude at terminal ***z*** of device ***d***. (Unit: decibels.)

ii(*d*,*n*) — Imaginary component of the complex current at node ***n*** of device ***d***.

ii*z*(*d*) — Imaginary component of the complex current at terminal **z** of device ***d***.

im(*d,n*) Current magnitude at node ***n*** of device ***d***.

im*z*(*d*) Current magnitude at terminal ***z*** of device ***d***.

ip(*d,n*) Current phase at node ***n*** of device ***d***.

ip*z*(*d*) Current phase at terminal ***z*** of device ***d***.

ir(*d,n*) Real component of the complex current at node ***n*** of device ***d***.

ir*z*(*d*) Real component of the complex current at terminal ***z*** of device ***d***.

vdb(*n1*[[,]*n2*]) Voltage magnitude at node ***n1*** relative to node ***n2*** (Unit: decibels. Default reference node: ground.)

vi(*n1*[[,]*n2*]) Imaginary component of the complex voltage at node ***n1*** relative to node ***n2***. (Default reference node: ground.)

vm(*n1*[[,]*n2*]) Voltage magnitude at node ***n1*** relative to node ***n2***. (Default reference node: ground.)

vp(*n1*[[,]*n2*]) Voltage phase at node ***n1*** relative to node ***n2***. (Default reference node: ground.)

vr(*n1*[[,]*n2*]) Real component of the complex voltage at node ***n1*** relative to node ***n2***. (Default reference node: ground.)

'frequency()' AC frequency. Must be or be part of an expression enclosed by single quotes.

The *arguments* for noise analysis mode (**.print noise**) include *any* of the arguments for AC analysis mode *in addition* to the following:

dn(*d*[,*t*]) Output noise spectral density contributions corresponding to the noise sources associated with device ***d***. If the noise type ***t*** (see below) is not specified, then results for all applicable noise types are printed.

inoise Equivalent input noise spectral density magnitude. (Unit: $\text{volts} / \sqrt{\text{Hertz}}$.)

inoise(db) Equivalent input noise spectral density magnitude. (Unit: decibels.)

inoise(tot) Total input noise—the integral of the input noise spectral densities over the analysis frequency interval. (Unit: volts.)

onoise Output noise spectral density magnitude (Unit: $\text{volts} / \sqrt{\text{Hertz}}$.)

onoise(db) Output noise spectral density magnitude. (Unit: decibels.)

onoise(tot) Total output noise—the integral of the output noise spectral densities over the analysis frequency interval. (Unit: volts.)

transfer AC transfer function between input and output. As the frequency approaches zero, this value approaches the result from the **.tf** command. (Unit: volts⁄ampere, for transresistance; or no unit, for voltage gain.)

The units for the **.print noise dn(*d*, *t*)** command are $\text{volts} / \sqrt{\text{Hertz}}$ by default. However, this can be chabged to Volts^2/Hertz by use of the option **dnout** (page 287).

The noise types ***t*** available for the **.print noise dn(*d*, *t*)** command vary according to the device type (BJT, Diode, JFET, etc.) as shown in the following tables:

BJT (Gummel-Poon) Noise Types

noise type	***description***
FN	Flicker noise due to base current
IB	Shot noise due to base current.
IC	Shot noise due to collector current.
RB	Thermal noise due to base resistance.
RC	Thermal noise due to collector resistance.
RE	Thermal noise due to emitter resistance.
RX	Transresistance from flicker noise source to output.
TOT	Total device output noise.

BJT (VBIC) Noise Types

noise type	***description***
IBE	Base-Emitter shot noise
IBEFN	Base-Emitter flicker noise
IBEP	Parasitic base-emitter shot noise
IBEPFN	Parasitic base-emitter flicker noise
ICCP	Parasitic base-collector shot noise
ITZF	Forward transport current shot noise
RBI	Thermal noise due to intrinsic base resistance
RBP	Thermal noise due to parasitic base resistance
RBX	Thermal noise due to extrinsic base resistance
RCI	Thermal noise due to intrinsic collector resistance
RCX	Thermal noise due to extrinsic collector resistance
RE	Thermal noise due to emitter resistance
RS	Thermal noise due to source resistance
RX	Transresistance from flicker noise source to output.
TOT	Total device output noise.

Diode Noise Types

noise type	***description***
FN	Flicker noise
ID	Shot noise.
RX	Transresistance from flicker noise source to output.
TOT	Total device output noise.

JFET and MESFET Noise Types

noise type	***description***
FN	Flicker noise.
ID	Thermal noise due to channel.
RD	Thermal noise due to drain resistance.
RG	Thermal noise due to gate resistance.
RS	Thermal noise due to source resistance.
RX	Transresistance from flicker noise source to output.
TOT	Total device output noise.

MOSFET Noise Types

noise type	***description***
FN	Flicker noise.
ID	Thermal noise due to channel.
RD	Thermal noise due to drain resistance.
RG	Thermal noise due to gate resistance.
RS	Thermal noise due to source resistance.
RX	Transresistance from channel or flicker noise source to output.
TOT	Total device output noise.

Examples

```
.print tran in out i1(r2) id(M2)
```

Prints transient analysis results: the voltages at nodes **in** and **out** and the currents into terminal **1** of device **r2** and the **drain** terminal of device **M2**.

```
.print dc I10/in, I10/out, I11/in, I11/out
```

Prints DC analysis results at various subcircuit nodes.

```
.print ac "acdata" im(M2,g1) vm(out) vdb(out)
```

Writes AC analysis results to output file **acdata**: the magnitude of the current flowing into node **g1** of device **M2**, the magnitude of the voltage at node **out**, and the same magnitude expressed in decibels.

```
.print noise inoise transfer dn(mn1) onoise(tot)
```

Prints noise analysis results: the equivalent input noise spectral density, the input/output transfer function, all noise information corresponding to device **mn1**, and the total output noise.

```
.print tran 'v(out)*sin(time()*sf)'
```

Prints the transient value of an expression involving the voltage at node **out**, the simulation time **time()**, and parameter **sf** (defined elsewhere with a **.param** command).

```
.print tran diff<V>='v(2)-v(1)'
```

Prints the transient value of an expression subtracting the voltage at node **1** from the voltage at node **2**. The string **diff** is used as a column heading with the letter V as a unit designation.

```
.print
```

Prints all node voltages and voltage source currents for all analyses to a text file.

```
.print tran
```

Prints all node voltages and voltage source currents for transient analysis to a text file.

```
.print v(n*)
```

Prints the voltages for all nodes whose name begins with the letter ’n’.

```
.print i[12](m*) i?(q*)
```

Prints the drain and gate currents (terminals 1 and 2) for every MOSFET device, and each terminal current for every BJT.

Device State variables are not available for all types of analysis or for all devices. In general, the state plots are only relevant to DC and transient analysis.

The format of the device state plot request is always ***state(d)***, where ***state*** is the state data identifier, and ***d*** is the device name.

The device state data which is available for each device type is as follows:

BJT (Gummel-Poon) Device State printout identifiers

state identifier	*description*
cap_be	cbe capacitance
cap_ibc	internal base-collector capacitance
cap_sbc	csc/csb substrate-collector/substrate-base capacitance
cap_xbc	external base-collector capacitance
cbo	base current
cco	collector current
cexbc	base-collector equivalent current
cqbc	current due to the base-collector charge
cqbe	current due to the base-emitter charge
cqbx	current due to the base-internal base charge
cqcs	current due to the collector-substrate charge
g0	∂ic/∂vce
gm	∂ic/∂vbe
gpi	∂ib/∂vbe
gu	∂ib/∂vbc
isub	substrate current
qbc	base-collector charge
qbe	base-emitter charge
qbx	base-internal base charge
qcs	collector-substrate charge
rb	base resistance
rgn	operating region: -2=inverse, -1=saturation, 0=off, 1=on
vbc	base-collector voltage
vbci	rb and rc offset internal base-collector voltage
vbe	base-emitter voltage
vbei	rb and re offset internal base-emitter voltage
vsub	substrate voltage

BJT (VBIC) Device State printout identifiers

state identifier	*description*
cbco	parasitic Base-Collector overlap capacitance (fixed)
cbeo	parasitic Base-Emitter overlap capacitance (fixed)
cqbc	Base-Collector charge current
cqbco	currents from Cbco charge
cqbcp	currents from Cbcp charge
cqbcx	currents from Cbcx charge
cqbe	Base-Emitter charge current
cqbeo	currents from Cbeo charge
cqbep	currents from Cbep charge
cqbex	currents from Cbex charge
cqcxf	currents from Ccxf charge
flxf	Excess phase circuit flux
ibc	intrinsic Base-Collector current
ibcp	parasitic Base-Collector current
ibe	intrinsic Base-Emitter current
ibep	parasitic Base-Emitter current
ibex	extrinsic Base-Emitter current
igc	weak avalanche current
irbi	intrinsic Base resistor modulated current
irbp	parasitic Base resistor modulated current
irbx	external Base resistor current
irci	intrinsic Collector resistor modulated current
ircx	external Collector resistor current
ire	external Emitter resistor current
ith	thermal (heat generation) source, power dissipation
itxf	forward transport current, with excess phase
itzf	forward transport current, zero phase
itzr	reverse transport current, zero phase
ixxf	forward transport current, with excess phase
ixzf	forward transport current, with excess phase
qbc	Base-Collector charge
qbco	parasitic Base-Collector charge (depletion)

BJT (VBIC) Device State printout identifiers

state identifier	*description*
qbcp	parasitic Base-Collector charge (depletion)
qbcx	parasitic Base-Collector charge (depletion)
qbe	Base-Emitter charge
qbeo	parasitic Base-Emitter charge (depletion and diffusion)
qbep	parasitic Base-Emitter charge (depletion and diffusion)
qbex	extrinsic Base-Emitter charge (depletion)
qcxf	Excess phase circuit capacitance
rbi	intrinsic Base resistance (modulated)
rbip	parasitic Base resistance (modulated)
rbx	extrinsic Base resistance (fixed)
rci	intrinsic Collector resistance (modulated)
rcx	extrinsic Collector resistance (fixed)
re	Emitter resistance (fixed)
rgn	operating region
rs	Substrate resistance (fixed)
vb	Base voltage
vbc	Base-Collector voltage
vbci	Rb and Rc offset internal Base-Collector voltage
vbe	Base-Emitter voltage
vbei	Rb and Re offset internal Base-Emitter voltage
vbi	Bi internal Base voltage
vbp	Bp parasitic Base voltage
vbx	Bx external Base voltage
vc	Collector voltage
vci	Ci internal Collector voltage
vcx	Cx external Collector voltage
ve	Emitter voltage
vei	Ei internal Emitter voltage
vs	Substrate voltage
vsi	Si internal Substrate voltage

Capacitor Device State printout identifiers

state identifier	*description*
ceff	effective capacitance
curr	current
dq	∂q/∂v
q	charge
volt	voltage potential

f element (CCCS) Device State printout identifiers

state identifier	*description*
curr	source current
di or di1	derivative of source current w.r.t. first control
di2	derivative of source current w.r.t. second control
di3	derivative of source current w.r.t. third control
volt	voltage potential across the CCCS

h element (CCVS) Device State printout identifiers

state identifier	*description*
curr	source current
dv or dv1	derivative of source voltage w.r.t. first control
dv2	derivative of source voltage w.r.t. second control
dv3	derivative of source voltage w.r.t. third control
volt	voltage potential across the CCVS

Diode Device State printout identifiers

state identifier	*description*
c	total diode capacitance
curr	current through the diode
di	∂i/∂v
dq	∂q/∂v
gd	conductance
id	current, excluding the series resistor

Diode Device State printout identifiers

state identifier	*description*
ir	current through the series resistor
qd	charge
rgn	operating region: -1=breakdown, 0=reverse, 1=forward
vd	voltage potential, excluding the series resistor
volt	voltage across the diode
vr	voltage across the series resistor

Inductor Device State printout identifiers

state identifier	*description*
curr	current through the inductor
ic	current through the component capacitor
ir	current through the component resistor
leff	effective inductance
vc	voltage across the component capacitor
volt	voltage across the inductor
vr	voltage across the component resistor

MOSFET Device State printout identifiers

state identifier	*description*
betaeff	effective beta
cap_bd	bulk-drain capacitance
cap_bs	bulk-source capacitance
cbdbo	∂Qb/∂Vd
cbdo	DC drain-bulk diode current
cbgbo	∂Qb/∂Vg
cbsbo	∂Qb/∂Vs
cbso	DC source-bulk diode current
cddbo	∂Qd/∂Vd
cdgbo	∂Qd/∂Vg
cdo	DC drain current
cdsbo	∂Qd/∂Vs
cgdbo	∂Qg/∂Vd

MOSFET Device State printout identifiers

state identifier	*description*
cggbo	∂Qg/∂Vg
cgsbo	∂Qg/∂Vs
cqb	current due to the intrinsic bulk charge
cqd	current due to the intrinsic drain charge
cqg	current due to the intrinsic gate charge
cqs	current due to the intrinsic source charge
deltal	channel length modulation
gammaeff	effective gamma
gbdo	Conductance of the drain diode
gbso	conductance of the source diode
gdso	DC drain-source transconductance
gmbso	DC substrate transconductance
gmo	DC gate transconductance
qbi	intrinsic bulk charge
qbd	bulk-drain diode charge
qbs	bulk-source diode charge
cqbd	current due to bulk-drain diode charge
cqbs	current due to bulk-source diode charge
qdi	intrinsic drain charge
qgi	intrinsic gate charge
qsi	intrinsic source charge
rgn	operating region: -1=subthreshold, 0=linear, 1=saturation
ueff	effective mobility
vbs	bulk-source voltage
vbsi	Rs offset internal bulk-source voltage
vds	drain-source voltage
vdsat	saturation voltage
vdsi	Rd and Rs offset internal drain-source voltage
vfbeff	effective Vfb
vgs	gate-source voltage
vgsi	Rs offset internal gate-source voltage
vth	Threshold voltage

Resistor Device State printout identifiers

state identifier	*description*
cap1	capacitance of the first capacitor
cap2	capacitance of the second capacitor
curr	current through the resistor
di	∂I/∂V
g	conductance
ic1	current through the first capacitor
ic2	current through the second capacitor
qc1	charge of the first capacitor
qc2	charge of the second capacitor
r	effective resistance
vc1	voltage across the first capacitor
vc2	voltage across the second capacitor
volt	voltage across the resistor

g element (VCCS) Device State printout identifiers

state identifier	*description*
curr	VCCS source current
di or di1	derivative of source current w.r.t. the first control
di2	derivative of source current w.r.t. the second control
di3	derivative of source current w.r.t. the third control
dq or dq1	derivative of source charge w.r.t. the first control
dq2	derivative of source charge w.r.t. the second control
dq3	derivative of source charge w.r.t. the third control
q	VCCS source charge
volt	voltage across the VCCS

e element (VCVS) Device State printout identifiers

state identifier	*description*
curr	VCVS source current
dv or dv1	derivative of source voltage w.r.t. the first control
dv2	derivative of source voltage w.r.t. the second control

e element (VCVS) Device State printout identifiers

state identifier	*description*
dv3	derivative of source voltage w.r.t. the third control
volt	voltage across the VCVS

.probe

Reports simulation results in binary format.

- Works exactly as **.print**, except that output is binary instead of text.
- If ***arguments*** are given, the ***mode*** must be specified.
- The file specified with the **.options** (page 95) **probefilename** command becomes the default for the **.probe** output file. If output filename is not specified, T-Spice uses an ASCII output file name with a **.dat** extension.

Syntax

```
.probe [mode] ["filename"] [arguments]
```

mode	Analysis type (see below). If *mode* is omitted, the **.probe** command applies to all analysis types.
filename	Specifies the binary output filename. The default extension is **.dat**. Must be enclosed by double quotes.
arguments	Specifies plot variables to be included in the output file. If ***arguments*** is omitted, T-Spice includes all node voltages and voltage source currents in the output. The format and types of ***arguments*** are the same as for the **.print** (page 108) command.

mode is one of the following:

tran	Print results from transient analysis.
dc	Print results from DC transfer analysis and DC operating point analysis.
ac	Print results from AC analysis.
noise	Print results from noise analysis.

Examples

```
.probe
```

Saves all node voltages and voltage source currents for all analyses to a binary file.

```
.probe tran
```

Saves all node voltages and voltage source currents for transient analysis to a binary file.

```
.probe tran v(2)
```

Saves the transient voltage at node **2** to a binary file.

```
.probe tran Output<v> = 'v(2) - v(1)'
```

Saves the transient value of an expression subtracting voltage at node **1** from voltage at node **2** and prints it in a column with the heading **Output**.

.protect / .unprotect

The **.protect** and **.unprotect** commands are used for temporarily turning input file echoing off and on. Input file echoing is initially turned on with the command **.option echo**. Subsequent use of **.protect** will turn off the echoing, until the **.unprotect** command is encountered. In this manner sensitive data, such as model libraries, can be protected from distribution, or excessive amounts of netlist echoing can be trimmed down.

Syntax

```
.protect

...

.unprotect
```

.save

Saves bias point information to a file. All of the non-internal node voltage values will be saved to the file using either the **.ic** or the **.nodeset** T-Spice command syntax. Subsequent simulations may use the **.load** command to read the file and execute the **.nodeset** or the **.ic** commmands.

Syntax

```
.save [file=filename] [type=ic | nodeset] [time=time]
```

filename Name of the file to be written. If the **file** parameter is not entered, then the filename is derived from the simulation output filename, with a **.ic** file extension.

type Denotes the type of node initialization command which will be written to the file for each non-internal circuit node. Either **ic**, to have .ic commands generated, or **nodeset**, for .nodeset commands. (*Default*: **nodeset**)

time The transient analysis time at which the bias information should be saved. Ordinarily, the **.save** command saves the DC operating point bias information. If a **time** parameter has been specified, and the simulation performs a transient analysis, then the bias at the specified timepoint will be saved.

Examples

```
.save type=ic time=50n
```

.savebias

Saves bias point information to a file. Node voltage values will be saved to the file using either the **.ic** or the **.nodeset** T-Spice command syntax. Subsequent simulations may use the **.load** command to read the file and execute the **.nodeset** or the **.ic** commmands.

Note: The .**save** and the .**savebias** commands perform essentially the same task, but use a different syntax and have different options.
.**save** is provided for HSPICE command compatibility, while .**savebias** provides PSPICE command compatibility.
The *saved* file from either command can be loaded into a subsequent simulation using the .**load** command.

Syntax

```
.savebias filename [op | dc | tran] [alter = alternum] [dc = dcvalue]
   [step = stepvalue] [temp = temperature] [ time = timevalue [ repeat ] ]
   [ic | nodeset] [internal] [nosubckt]
```

filename	Name of the file to be written.
op | dc | tran	Indicates the type of analysis for which voltage values will be saved. (Default: **op**)
alternum	Identifies the alter block index number for which data should be saved. (Default: 0)
dcvalue	Identifies the DC sweep value for which data should be saved. (Default: all)
stepvalue	Identifies the **.step** value for which data should be saved. (Default: all)
temperature	Identifies the **.temp** temperature for which data should be saved. (Default: all)
timevalue	Identifies the transient timepoint at which data should be saved. (Default: 0)
repeat	For transient timepoint saving, indicates that the output file should repeatedly be overwritten for each timepoint which is an integral multiple of ***timevalue***.
ic	Indicates that the node initialization command which is written should be the **.ic** command.
nodeset	Indicates that the node initialization command which is written should be the **.nodeset** command.
internal	Indicates that all internal node values should be included in the output. Internal nodes are those nodes which were not in the input circuit, but were automatically generated internal to devices.
nosubckt	Indicates that only the top level circuit nodes, excluding subcircuit nodes, should be written.

Examples

```
.savebias ring.ic tran ic time=100n repeat

.savebias dc alter=2 dc=2.5 temp=75 internal
```

.step

Performs a parametric sweep of a sweep variable, performing all analyses in the input file for all parameter values in the sweep.

The **.step** command produces a separate output section for each parameter value of the sweep. For example, an input file with **.step** and **.tran** (page 137) produces one transient analysis output section for each parameter value in the sweep. In addition, all **.macro /.eom** (page 77) results are plotted as traces with the swept variable as the *x*-axis. The output format for this is similar to that of the **.dc** (page 60) command.

Note: The **.step** command can be abbreviated to **.st**.

Syntax

.step *sweep* [[**sweep**] *sweep* [[**sweep**] *sweep*]]

where ***sweep*** is in one of the following formats:

[**lin**] *variable start stop inc*

or

dec|**oct** *variable start stop npoints*

or

variable **lin**|**dec**|**oct** *npoints start stop*

or

variable **list** *value* [*value* [**...**]]

or

list *variable value* [*value* [**...**]]

or

variable **poi** *npoints* [*value* [**...**]]

or

data=*dataname*

or

monte=*mcruns* [**seed=***seedval*]

or

optimize=opt*name* **results=***measname* [*measname* [**...**]] **model=***optmodelname*

variable specifies the parameter whose value is to be swept. It is one of the following:

temp Specifies a temperature sweep.

param *paramname* Sweeps a global parameter named ***paramname*** defined using the **.param** command.

source *sourcename* Sweeps the DC value of a voltage or current source value named ***sourcename***.

[**modparam**] ***parname***(***modelname***) Sweeps the value of model parameter *parname* for the device model ***modelname***.

paramname Sweeps a global parameter (as with **param**) or DC source value (as with **source**). T-Spice first looks for a matching **.param** parameter, and then for a source name.

Other parameters include the following:

start Specifies the beginning of a linear or logarithmic sweep.

stop Specifies the end of a linear or logarithmic sweep.

inc Specifies the increment for a linear sweep.

npoints Specifies the total number of points for a linear or **poi** sweep, or the number of points per decade or octave for a logarithmic sweep.

value Specifies a single value which ***variable*** takes on for one step of the sweep.

mcruns Specifies the number of runs to be performed for Monte Carlo analysis.

seedval An integer specifying a seed for initializing the random number sequence for Monte Carlo analysis. If ***seedval*** is negative, T-Spice uses the system clock to generate a different seed each time the simulation is run. A ***seedval*** of zero is equivalent to not specifying a seed value at all.

dataname Specifies the name of a **.connect** (page 57) statement to be used for the sweep. The column names in the **.data** statement must correspond to global **.param** (page 102) parameters. For each sweep step, those parameters are assigned the values found in one row of data produced by the **.data** statement.

sweep Specifies that analysis be performed for all parameter values of the sweep and indicates the beginning of the next nested sweep variable specification. The **sweep** keyword can be omitted if the previous sweep is not of the **list** or **poi** type or if one of the keywords **lin**, **dec**, **oct**, **list**, **poi**, **temp**, **param**, **source**, or **modparam** follows immediately.

lin Specifies a linear sweep.

dec Specifies a logarithmic sweep by decades.

oct Specifies a logarithmic sweep by octaves.

list Specifies a sweep over a list of values (P-Spice compatible syntax).

poi
Specifies a sweep over a list of values (HSPICE compatible syntax).

data
Specifies a sweep defined using a **.data** statement.

monte
Specifies a Monte Carlo sweep. For each Monte Carlo run, random circuit parameter values are generated from probability distributions. A Monte Carlo sweep must be the outermost sweep if sweeps are nested.

optimize
Specifies an optimization sweep. During an optimization sweep, T-Spice runs many analyses in an attempt to optimize a circuit performance objective. The user may specify a set of parameters to be varied and a set of measurements to be included in the optimization goal. Optimization sweeps may not be nested within other optimization sweeps. For further information on setting up an optimization run, see Optimization on page 493.

optname
Selects a set of parameters to be varied in an optimization run. The parameters to be optimized are specified using **.param** (page 102) with a matching **optname**.

results=***measname***
Specifies circuit measurement results to be used for defining an optimization goal. Each ***measname*** refers to a **.macro /.eom** (page 77) command of the same name and contributes to the optimization goal. The complete optimization goal is the RMS of all measurements listed. For further information on specifying circuit measurement results, see Defining Optimization Goals on page 494.

model=***optmodelname***
Specifies an optimization algorithm model name. It is matched with a **.model** (page 85) statement of type **opt** and name ***modelname***. That **.model** statement specifies parameters for the optimization algorithm. For further information on specifying an optimization algorithm, see Optimization on page 493.

Examples

```
.step vin 0 5 0.1
```

sweeps the DC value of voltage source **vin** from **0** to **5V** with **0.1V** increments.

```
.step lin param ml 2 3 0.5 sweep vdd 3 5 0.1
```

performs a nested linear sweep of the parameter **ml** and the voltage source **vdd**.

```
.step list temp 0 27 100 150 -50
```

sweeps the circuit operating temperature over the five values listed.

```
.step optimize=opt1 results=bandwidth,delay model=optmod
.param p1=opt1(1e-3,1e-5,1) p2=opt1(150,100,200)
.model optmod opt level=1 itropt=40
.measure ac bandwidth trig vm(out) val=0.5 cross=1
+                     targ vm(out) val=0.5 cross=2
+                     goal=2kHz
.measure tran delay when v(1)=2.5 goal=10ns
```

invokes an optimization of parameters **p1** and **p2**. T-Spice will attempt to find values for **p1** and **p2** which result in a bandwidth of 2 kHz and a delay of 10 ns. An AC and a transient analysis would be performed for each optimization function evaluation.

.subckt

Defines a hierarchical set of devices and nodes to be used repeatedly in a higher-level circuit.

- Subcircuits are replicated by means of the instance (**x**) statement.
- When invocations of the following commands appear within subcircuit definitions and refer to nodes inside the subcircuit, the commands are executed for each instance of the node: **.acmodel** (page 54), **.ic** (page 70), **.macro /.eom** (page 77), **.nodeset** (page 89), **.noise** (page 90), **.print** (page 108), and **.probe** (page 122), and **.tf** (page 136).
- Node and device names have local scope in subcircuits unless global node names (defined elsewhere with the **.global** (page 68) command) are used.
- Subcircuit blocks cannot be nested: after one **.subckt** command, the **.ends** (page 65) command must appear before another **.subckt** command can be used.

Syntax

```
.subckt name node1 [node2 ...] [parameter=X ...]
subcircuit
.ends
```

name Name of subcircuit.

node1 node2 Nodes used as “external” connections to the subcircuit.

parameter Parameter(s), with default value(s) assigned. ***X*** can be a number or an expression. Subcircuit parameters have local scope. Parameters can be written in any order in both definition and instances. Parameter values specified in the definition are used as defaults when not specified in instances. Within the definition, parameter values are referenced (in place of numbers) by enclosing their names in single quotes. Alternatively, the **.param** (page 102) command may be used within the definition, with the same results. Parameters created outside the definition with the **.param** (page 102) command may be used inside the definition, but an assignment made with the **.subckt** command to an externally defined parameter always overrides its external value.

subcircuit Subcircuit definition (may be multiple lines).

Examples

```
.subckt inv in out Vdd length=1.25u nwidth=2u pwidth=3u
mt1 out in GND GND nmos l='length' w='nwidth'
mt2 out in Vdd Vdd pmos l='length' w='pwidth'
c2 out GND 800f
.ends inv
```

This subcircuit could be instanced as follows:

```
xinv1 a1 a2 Vdd inv nwidth=4u pwidth=6u
```

.table

Links a table reference name (used by simulated devices) with external charge and current table filenames.

The table which is imported from the named file can define the behavior of a standard device, such as a MOSFET, resistor, diode, etc., and takes the place of the ***modelname*** (from a **.model** definition) in the device declaration.

The table data may also be used to define a generic "black box" device which is created by the **x*name*** method of instantiating a subcircuit. When the **.table** and the **x*name*** commands are used in combination to define a generic table-based device, the **.table** model takes the place of a **.subckt** subcircuit definition. You use the **.table** command instead of the **.subckt** command to define your device characteristics.

Syntax

```
.table tablename current charge
```

tablename Table reference name.

current File containing steady-state current table. The file should have extension **.f** (binary) or **.ftx** (ASCII). If the referenced filename or path contains a space, enclose the entire path in single or double quotation marks.

charge File containing charge table. The file should have extension **.q** (binary) or **.qtx** (ASCII). If the referenced filename or path contains a space, enclose the entire path in single or double quotation marks.

Examples

MOSFET table example:

```
.table nmos1x3 n1x3.ftx n1x3.qtx
```

Establishes a relationship between the table reference name **nmos1x3** and the table files **n1x3.ftx** and **n1x3.qtx**. The input file is searched for any device statements—MOSFET (**m**) statements or **xname** commands—which include the table reference name **nmos1x3**. The corresponding devices use the charge and current tables in **n1x3.ftx** and **n1x3.qtx**.

A table-based MOSFET device can then be created by (for example) :

```
mnmos1x3 n1 n5 GND GND nmos1x3
```

Inverter table example:

```
xinv in out invtable
.table invtable invcirc.ftx invcirc.qtx
```

Illustrates how a CMOS inverter can be defined as a macromodel. Rather than explicitly constructing a description of the inverter with its component transistors, the circuit description names the inverter's input and output nodes. The input/output behavior of the inverter is approximated by current (**invcirc.ftx**) and charge (**invcirc.qtx**) tables that are referenced with the name **invtable**.

Resistor table example:

```
v1 1 0 1000
.table an_ftx_qtx_pair resistor.ftx resistor.qtx
xr1 1 0 an_ftx_qtx_pair
.op
```

where **resistor.ftx** is:

```
1 2 1 0
-1 +1
-1e-3
+1e-3
```

and **resistor.qtx** is:

```
1 2 1 0
-1 +1
0
0
```

Illustrates a resistor defined as a macromodel where the circuit description names the resistor's input and output nodes. The input/output behavior of the resistor is approximated by a table referenced with the tablename **an_ftx_qtx_pair**

.temp

Specifies the temperatures at which the circuit is to be simulated.

- Changing the temperature affects the behavior of diode, resistor, BJT, JFET, MESFET, and MOSFET models. It may also affect the behavior of user-defined external models.
- The **.temp** command has no effect on external tables, which should be regenerated to reflect the new temperature.

Syntax

```
.temp temperature [temperature [temperature [...]]]
```

temperature	Temperature. (Unit: °C. Default: 25.)

.tf

Computes and reports the value of the small-signal DC transfer function between the specified output and input, and the corresponding input and output resistances, at the DC operating point.

- The **.tf** command automatically performs (but does not report the results from) a DC operating point calculation.
- Results are reported under the heading **SMALL-SIGNAL TRANSFER FUNCTION** (to the specified output file or in the Simulation Window).
- The transfer function value corresponds to a voltage (V /V) or current (I /I) *gain*, a *transconductance* (I/V), or a *transresistance* (V/I).

Syntax

```
.tf arguments source
```

arguments Any arguments appropriate for the **.print** (page 108) **dc** command.

source Voltage or current input source.

Examples

```
.tf i(mb1,out1) ii1
```

Computes transfer function results between node **out1** of device **mb1** and current source **ii1**.

.tran

Performs large-signal time-domain (transient) analysis of the circuit to determine its response to initial conditions and time-dependent stimuli.

- The time step is adaptively varied throughout the simulation to ensure accuracy.
- Results for nodes selected by the **.print tran**, **.probe tran**, and **.measure tran** commands will be output for every time step, unless otherwise specified by the **.options prtdel** command. For additional information on these commands, see **.print** (page 108), **.probe** (page 122), **.macro /.eom** (page 77) and **.options** (page 95).

Syntax

```
.tran[/mode] S L [start=A] [UIC] [sweep sweep]
```

mode Analysis mode (see below). This parameter must immediately follow the keyword **.tran** and be preceded by a slash (/).

S Maximum time step allowed. By default, the time step is dynamically adapted to resolve the output values. (Unit: seconds.)

L Total simulation time. (Unit: seconds.)

A Output start time. Execution of the **.print tran** command will not start until this time. (Unit: seconds. Default: 0.)

UIC Instructs T-Spice to skip the DC operating point analysis for determining the **time=0** circuit state. Instead, only the initial conditions specified using **.ic** commands are used to set the **time=0** voltages. Voltages which cannot be determined using **.ic** commands are set to zero.

sweep For a description of the syntax for this field, see **.step** (page 128).

mode takes one of the following values:

op Performs a DC operating point calculation before simulation to determine initial steady-state node voltages. The commands **.nodeset** (page 89) or **.ic** (page 70) can be used to impose initial conditions.

powerup Performs a “powerup” simulation. All nodes are at the same potential at time zero, and the voltage sources are ramped gradually to their final values.

preview Steps through the input signals without simulating the circuit. Input waveforms will be reported as specified by the **.print** (page 108) **tran** command.

If ***mode*** is not specified, T-Spice first performs a DC operating point analysis, without printing the DC operating point analysis results.

sweep indicates the beginning of the next nested sweep variable specification. The **sweep** keyword can be omitted if the previous sweep is not of the **list** or **poi** type or if one of the keywords **lin**, **dec**, **oct**, **list**, **poi**, **temp**, **param**, **source**, or **modparam** follows immediately.

Using the ***sweep*** option with **.tran** or **.ac** (page 51) causes that analysis to be performed for all parameter values of the sweep. It is equivalent to **.step** (page 128), except that it applies only to one analysis command, while **.step** applies to all analysis commands in the input file. If ***sweep*** is specified on an analysis command and **.step** is present, the ***sweep*** sweep is nested inside the **.step** sweep. The ***sweep*** parameter may be used to specify a parametric sweep, Monte Carlo analysis, or optimization.

Examples

```
.tran 0.5n 100n
```

Defines a transient simulation lasting 100 nanoseconds, using time steps of at most 0.5 nanosecond. By default, a DC operating point calculation will first be performed to define a starting condition.

```
.tran/preview 4n 4000n
```

The input waveforms are reported for 4000 nanoseconds; the rest of the circuit is ignored.

```
.tran 1ns 100ns
```

Specifies a maximum time step of 1 nanosecond and a total simulation time of 100 nanoseconds.

```
.tran/powerup 1ns 100 ns
```

Specifies a powerup simulation with no operating point computation.

```
.tran 1ns 100 ns start=50ns
```

Produces output starting at time 50 nanoseconds.

```
.tran 1n 100n sweep temp list 0 27 100 150 -50
```

Performs five transient analysis runs, one for each temperature listed. The keyword **temp** specifies the sweep *variable*, as defined in **.step** (page 128).

```
.tran 1n 400n sweep temp -50 150 50
```

This performs five transient analyses at temperatures -50, 0, 50, 100, and 150 degrees Celsius.

```
.tran 0.5u 100u sweep monte=20
```

This performs 20 transient simulations as part of a Monte Carlo analysis. The keyword **monte** defines one of the many **sweep** options described in **.step** (page 128)). For each of the 20 transient analyses, values are randomly chosen for circuit variables, which are assigned probability distributions according to the specified **Monte Carlo Parameters** (page 103).

For a demonstration of Monte Carlo analysis in T-Spice, see Example 2: Monte Carlo Analysis on page 491.

```
.tran 1n 200n sweep temp list 0 25 75 150
```

This example performs four transient analysis runs at temperatures 0, 25, 75, and 150 degrees Celsius.

.vector

Names a bus and specifies how many bits the bus will contain.

- The bus is connected to a vector-valued current or voltage source, defined by a **i** or **v** statement with the **bus** keyword.
- The input source generates signals composed of bit strings of the length specified in the **.vector** command.

Syntax

```
.vector bus {node1 [[,] node2 ...]]}
```

bus Bus name.

node1 node2 Input nodes. If there are ***n*** nodes in a bus, the rightmost ***n*** bits of the input waveform (a binary number) are assigned one by one to these nodes. The last-named (***n***th) node is assigned the least-significant (rightmost) bit; the (***n***–1)th node is assigned the next bit to the left; and so on. Extra bits are discarded. Extra nodes are set to zero.

Examples

```
.vector bus1 {b7 b6 b5 b4 b3 b2 b1 b0}
```

Defines a bus **bus1** and lists its eight input nodes. The input waveform to these nodes is specified as some number or numbers, convertible to a binary string with at least eight bits, in the accompanying voltage or current source statement.

.vrange

Sets the voltage range used for all table dimensions of a specific device type.

- Changing the voltage range of a table does not change the number of points in each dimension of the table. Increasing the voltage range will compromise accuracy unless the gridsize is also increased.

Syntax

```
.vrange type V
```

type	Type of device (see below).
V	Maximum table voltage (volts). Only one value is required; tables range from –***V*** to ***V***.

type is one of the following:

diode	DIODE
jfet	JFET
mes	MESFET
mos	MOSFET

Examples

```
.vrange mos 15
```

Causes MOSFET tables to range, for the purposes of interpolation, from –15 to +15 V in all dimensions. (Values outside the range are extrapolated.)

6 Device Statements

Introduction

This chapter documents the *device statements* of the T-Spice circuit description language.

The *device types* are listed in alphabetical order; each type is associated with a *key letter* (in parentheses). Many statements have “options,” which branch to different modes, and “arguments,” which indicate expressions, nodes, or devices to be used. In the input file, a device statement must begin with its key letter in the first column of the line containing it (no leading spaces). Options and arguments must be separated by spaces or new lines (with line continuation).

Syntax sections in this documentation follow these conventions:

- ***Italics*** indicate variables to be replaced by actual names, numbers, or expressions.
- Curly brackets **{}** indicate alternative values for the same option or argument.
- Square brackets **[ ]** enclose items that are *not required*.
- Vertical bars **|** separate alternative values for the same option or argument.
- Ellipses **…** indicate items that may be repeated as many times as needed.

These characters are *not* typed in the input file. All other characters are typed as shown.

For more information, see Input Conventions on page 42 and Simulation Commands on page 50.

BJT (q)

A transistor with up to four terminals: collector, base, emitter, and (optional) substrate. (BJT stands for *bipolar junction transistor*.)

Several types of bipolar models are supported in T-Spice:

SPICE Gummel-Poon model (level 1)
Vertical Bipolar Inter-Company (VBIC) (level 9)
Philips MEXTRAM (levels 6, 503, and 504)
Philips Modella (level 10 and 500)

The substrate is optional so that both discrete and IC BJTs may be modeled correctly.

Syntax

```
qname collector base emitter [substrate] model [[area=]A] [areab=B]
    [areac=C] [M=M] [SCALE=S] [tables=T]
```

Parameter	*Symbol*	*Description*	*Default*
name		BJT name	
collector		Collector terminal	
base		Base terminal	
emitter		Emitter terminal	
substrate		Substrate terminal	
model		BJT model name. The model is specified elsewhere in the input file in the form **.model** name npn|pnp [parameters].	

The following device options are available for Gummel-Poon models. Note that the tables parameter default is determined by the global modelmode option, which defaults to direct mode (tables=0).

Parameter	*Symbol*	*Description*	*Default*
area	A	Area scale factor	1
areab	B	Base area scale factor	A
areac	C	Collector area scale factor	A
M	M	Multiplicity - the number of devices to be placed in parallel.	1
tables	T	Toggle internal tables. When internal tables are on, T-Spice will build a table of current and charge values to speed device evaluation.	1

VBIC device statements are different from the Gummel-Poon bipolars. The VBIC model does not contain any terms for explicitly defining geometry or junction areas.

```
qname collector base emitter [substrate] [tmode] model [M=M] [SCALE=S]
[tnodeout]
```

Instead, the device **SCALE** parameter is provided for linearly scaling the device currents and charges. For compatibility with Gummel-Poon devices, the VBIC device statement will also accept the **M** multiplicity factor as a synonym for the **SCALE** parameter.

Parameter	***Symbol***	***Description***	***Default***
scale	S	Scale factor	1

Examples

```
qout1 r32 r23 gnd sub1 npnmod
qout2 r32 r23 gnd sub1 npnmod area=2
```

The **area** factor scales the generated current; thus, **qout2** generates twice as much current as **qout1**.

Capacitor (c)

A two-terminal capacitor. A nonlinear capacitor can be created using the **g**- element with an expression and the **chg** keyword. See Nonlinear Capacitor on page 184.

Syntax

```
cname node1 node2 CapValue|C=CapValue[=C] [M=M] [scale=scale] [tc1=T1]
    [tc2=T2] [dtemp=dtemp]
```

.or

```
cname node1 node2 POLY c0 [c1 [...]] [M=M]
```

or

```
cname node1 node2 modelname [c=C] [M=M] [scale=scale] [tc1=T1] [tc2=T2]
    [dtemp=dtemp] [l=length] [w=width]
```

or

```
cname node1 node2 modelname C [T1 [T2 ]] [M=M] [scale=scale] [dtemp=dtemp]
    [l=length] [w=width]
```

Parameter	*Symbol*	*Description*	*Default*	*Units*
name		Capacitor name.		
node1		Positive terminal		
node2		Negative terminal.		
C	C	Capacitance.		F
POLY		Keyword indicating that the capacitance is a polynomial function.		
c0, c1, ...	C_0, C_1, …	Coefficients of the polynomial function for capacitance.		
M	M	Multiplicity - the number of devices to be placed in parallel.	1	
scale	S	Element scale factor	1	
T1	T_1	First temperature coefficient for capacitance.	0	1/deg
T2	T_2	Second temperature coefficient for capacitance.	0	(1/deg^2)
dtemp	D_{temp}	Difference between the capacitor and the circuit temperatures.	0	deg
l	L	Length of capacitor.		m
w	W	Width of capacitor		m

Equations

If the first syntax is employed the capacitance is calculated as

$$C = M \cdot S[1 + T_1(\Delta T) + T_2(\Delta T)^2] \cdot C_0 \tag{0.8}$$

where

$$\Delta T = T_{circuit} + D_{temp} - T_{nom} \tag{0.9}$$

where $T_{circuit}$ is set in **.temp** and T_{nom} in **.options tnom**.

If the second syntax is employed, capacitance is calculated as

$$C = c_0 + c_1 V + c_2 V^2 + \dots \tag{0.10}$$

where V denotes the voltage between ***node1*** and ***node2***.

If the third or fourth syntax is employed, there must be a matching **.model** (page 85).

Note: When the calculated capacitance is greater than 0.1 F, T-Spice issues a warning message.

Examples

```
cwire w1 gnd 82f
```

The example defines a capacitor with a value of 82 femtofarads. A common error is to omit the metric abbreviation on the capacitance value, which can lead to unexpected results.

```
cxx 1 0 poly 0.08 2.08 3.08
```

This example defines a capacitor **cxx** connected between nodes **1** and **0**. The capacitance of **cxx** is described as

$$C = 0.08 + 1.08V + 2.08V^2 + 3.08V^3 \tag{0.11}$$

where V is the voltage between nodes **1** and **0**.

A capacitor exhibiting polynomial dependence on its applied voltage can be modeled using the **POLY** keyword:

```
c1 n+ n- POLY c0 'c0*vcc'
```

The waveform for this capacitor is illustrated in Nonlinear Capacitor on page 184.

```
c1 10 20 capxx 0.02 1.5e-2 5.0e-4 dtemp=20
```

This example illustrates the fourth syntax. The capacitor is named **c1**. Its terminals are connected to nodes **10** and **20**. Its model name is **capxx**. It has two temperature coefficients, **tc1 = 1.5e-2** and **tc2 = 5.0e-4**. Its **dtemp = 20**. As the model name is **capxx**, the corresponding **.model** statement must also contain the word **capxx**.

Coupled Transmission Line (u)

A set of coupled transmission lines.

There is no limit (besides physical memory) on the number of transmission lines that can be coupled.

Syntax

```
uname in1 [in2 […]] in0 out1 [out2 […]] out0 model length=L [lumps=X]
   [lumptype=Y]
```

Parameter	*Symbol*	*Description*	*Default*	*Units*
name		Coupled transmission line name.		
in1 in2 ...		Input terminals (as many as needed).		
out1 out2 ...		Output terminals (as many as needed);.		
in0		Input reference terminal.		
out0		Output reference terminal.		
model		CPL coupling model name. The model is specified elsewhere in the input file in the form **.model** name cpl level=l [[r]={matrix}] [c]={matrix} [l]={matrix} [[g]={matrix}]		
length	L	Physical length.		m
lumps	X	Number of lumps used for iterative ladder circuit (ILC) expansion.	1	
lumptype		Type of lumps used for ILC expansion. Y is one of the following: 0 = “Gamma” type lumps 1 = “Tee” type symmetric lumps 2= “Pi” type symmetric lumps 3= Hybrid RGT lumps	3	1/deg

Examples

```
ufine in1 in2 in3 refin out1 out2 out3 refout
+ cplmod l=1m lumps=3 lumptype=1
```

Current Source (i)

A two-terminal ideal current supply.

Exponential, pulse, piecewise linear, frequency-modulated, sinusoidal, and customizable (vectorized) waveforms can be specified.

To specify a current source with an equation, use the **Voltage-Controlled Current Source (g)** (page 181) with an expression that may involve the **time()** function.

Syntax

```
iname node1 node2 [[DC] I] [AC M [P]] [waveform]
```

name	Voltage source name.
node1	Positive terminal—or bus named by an associated **.vector** command.
node2	Negative terminal.
I	DC level. (*Unit:* A)
M	AC Magnitude. (*Unit*: A)
P	AC Phase (*Unit:* degrees. *Default:* 0.)
waveform	Waveform identifier and parameters (see below).

DC, AC, and transient values can be specified independently and in any order.

waveform is one of the following:

Exponential Waveform

```
exp (Ii Ip [Dr [Tr [Df [Tf]]]])
```

Ii	Initial current. (*Unit:* amperes.)
Ip	Peak current. (*Unit:* amperes.)
Dr	Rise time delay. (*Unit:* seconds. *Default:* 0.)
Tr	Rise time constant. (*Unit:* seconds. *Default:* 0.)
Df	Fall time delay. (*Unit:* seconds. *Default:* 0.)
Tf	Fall time constant. (*Unit:* seconds. *Default:* 0.)

The formula used is:

$$I(t) = \begin{cases} I_i & 0 \le t \le D_r \\ I_i + (I_p - I_i)\left(1 - \exp\left(\frac{-(t - D_r)}{T_r}\right)\right) & D_r \le t \le D_f \\ I_i + (I_i - I_p)\left(1 - \exp\left(\frac{-(t - D_f)}{T_f}\right)\right) & D_f \le t \end{cases} \tag{0.12}$$

Pulse Waveform

```
pulse (Ii Ip [D [Tr [Tf [Pw [Pp]]]]]) [ROUND=RND]
```

Ii Initial current. (*Unit:* amperes.)

Ip Peak current. (*Unit:* amperes.)

D Initial delay. (*Unit:* seconds. *Default:* 0.)

Tr Rise time. (*Unit:* seconds. *Default:* time step from **.tran**.)

Tf Fall time. (*Unit:* seconds. *Default:* time step from **.tran**.)

Pw Pulse width. (*Unit:* seconds. *Default:* stop time from **.tran**.)

Pp Pulse period. (*Unit:* seconds. *Default:* stop time from **.tran**.)

RND Rounding half-interval. A corner at time *T* is replaced by a smoothly differentiable polynomial in the interval (*T*–***RND***,*T*+***RND***). The maximum ***RND*** is half the distance to the nearest neighboring corner. (*Default:* 0—no rounding.)

Note that beginning with version 11, T-Spice interprets rise time as the time to go from the initial voltage to the pulse voltage, regardless of which is larger.

Piecewise Linear Waveform

```
pwl (T1 I1 [T2 I2 ...])[ROUND=RND] [REPEAT[=Tr]] [TD=DELAY]
```

T1 T2 Time at corner 1, 2, and so on. (*Unit:* seconds.)

I1 I2 Current at corner 1, 2, and so on. (*Unit:* amperes.)

ROUND Rounding half-interval. A corner at time *T* is replaced by a smoothly differentiable polynomial in the interval (*T*–***RND***,*T*+***RND***). The maximum ***RND*** is half the distance to the nearest neighboring corner. (*Default:* 0—no rounding.)

REPEAT Starting time within the specified waveform for an infinite number of repetitions of the subwaveform. If ***Tr*** is not specified, the entire waveform repeats indefinitely (i.e., ***Tr***=0). ***Tr*** must be less than or equal to the duration of the waveform. Waveforms can only repeat if the start and end points match. If they do not match, the repeat option is ignored. The **REPEAT** keyword can be abbreviated to **R**.

TD — Time delay added to the beginning of the waveform. If you specify corners ***T1, T2,*** *etc*. and **TD=*DELAY***, then the defined current values will actually be applied at effective corner times ***T1+DELAY, T2+DELAY,*** *etc*.

Piecewise Linear Waveform File

```
pwlfile filename [ROUND=RND] [REPEAT[=Tr]] [TD=DELAY]
```

filename — Input file which contains the piecewise linear waveform definition in a series of **time, current** pairs, one per line.

ROUND — Same meaning as with **pwl** waveforms

REPEAT — Same meaning as with **pwl** waveforms

TD — Same meaning as with **pwl** waveforms

Frequency-Modulated Waveform

```
sffm (Io Ip [Fc [Xm [Fs]]])
```

Io — Offset current. (*Unit:* amperes.)

Ip — Peak current. (*Unit:* amperes.)

Fc — Carrier frequency (*Unit:* Hertz. *Default:* $1 / T$, where T is the stop time from **.tran**.)

Xm — Modulation index. (*Default:* 0.)

Fs — Signal frequency. (*Unit:* Hertz. *Default:* $1 / T$, where T is the stop time from **.tran**.)

The formula used is:

$$I(t) = I_o + I_p \cdot \sin[(2\pi \cdot F_c \cdot t) + Xm \cdot \sin(2\pi \cdot F_s \cdot t)] \qquad (0.13)$$

Sinusoidal Waveform

```
sin (Io Ip [Fr [De [Da [Ph]]]])
```

Io — Offset current. (*Unit:* amperes.)

Ip — Peak current. (*Unit:* amperes.)

Fr — Frequency. (*Unit:* Hertz. *Default:* $1 / T$, where T is the stop time from **.tran**.)

De — Delay time. (*Unit:* seconds.)

Da — Damping factor. (*Unit:* 1/seconds.)

Ph Phase advance. (*Unit:* degrees.)

The formula used is:

$$I(t) \;=\; I_o + I_p \cdot \sin\left(2\pi \cdot \left(F_r \cdot (t - t_d) + \frac{P}{360}\right)\right) \cdot \exp(-((t - t_d) \cdot D)) \qquad (0.14)$$

Vectorized Waveform

```
bit|bus ({pattern} [on=A] [off=Z] [delay=D] [pw=P] [rt=R] [ft=F] [lt=L]
    [ht=H]) [ROUND=RND]
```

pattern An expression consisting of one or more string or string-multiplier combinations (see below).

A On current (*Unit:* amperes. *Default:* 0.001.)

Z Off current. (*Unit:* amperes. *Default:* 0.)

D Delay time. (*Unit:* seconds. *Default:* 0.)

P Pulse width: ***P*** = ***R*** + T_A = ***F*** + T_Z, where T_A is the time during a pulse where the current is “on” (V = ***A***) and T***Z*** is the time during a pulse where the current is “off” (V = ***Z***). (*Unit:* seconds. *Default:* 10×10^{-9}.)

R Rise time. (*Unit:* seconds. *Default:* 1×10^{-9}.)

F Fall time. (*Unit:* seconds. *Default:* 1×10^{-9}.)

L Low time: ***L*** = ***F*** + T_Z, where T_Z is the time during a pulse where the current is “off” (V = ***Z***). (*Unit:* seconds. *Default:* 10×10^{-9}.)

H High time: ***H*** = ***R*** + T_A, where T_A is the time during a pulse where the current is “on” (V = ***A***). (*Unit:* seconds. *Default:* 10×10^{-9}.)

RND Rounding half-interval. A corner at time *T* is replaced by a smoothly differentiable polynomial in the interval (*T*–***RND***,*T*+***RND***). The maximum ***RND*** is half the distance to the nearest neighboring corner. (*Default:* 0—no rounding.)

A *bit* pattern consists of a set of numbers (possibly associated with multiplier factors) whose binary representations sequentially specify the “on”/“off” structure of the waveform. The pattern takes the form ***a*(*b*(*x*) *c*(*y*) …)**, where ***a***, ***b***, and ***c*** are the optional multiplier factors and ***x*** and ***y*** are the numbers.

A *bus* pattern consists of a set of numbers (possibly associated with multiplier factors) whose binary representations—“bit strings”—are grouped together as a waveform bus and treated as a single input. The length of the bit strings is specified by the **.vector** command. If there are *n* nodes in a vector, then T-Spice assigns the first *n* bits of each bit string to those nodes. Extra bits are discarded. If there are not enough bits, the highest-order bits are set to zero. The leftmost node name in the **.vector** command takes the most significant bit.

Numbers are specified on the device statement in binary, hexadecimal (suffixed by **h**), octal (suffixed by **o**), or decimal (suffixed by **d**) notation. (For decimal representations the number of lower-order bits to be collected is also given.)

Examples

```
i1 a b 4.5u AC 1.0m 0.0
```

i1 has a DC value of 4.5 microamps, an AC magnitude of 1 milliampere, and an AC phase shift of 0 degrees.

```
i2 n2 GND pwl (0n 0 100n 0 101n 5 300n 5 301n 0
+              500n 0 680n 5 700n 0 880n 5 900n 0)
```

i2 generates a **pwl** (piecewise linear) input: a single pulse followed by a pair of sawtooth cycles.

```
i3 n3 GND bit ({01010 11011} on=5.0u off=0.0 pw=50n rt=10n ft=30n)
```

i3 generates a **bit** input. Enclosed in braces **{ }** are two binary-valued five-bit patterns specifying the waveform. The two patterns alternate in time. The **on** current value is 5.0 microamps; the **off** current value is zero. The pulse width (**pw**), 50 nanoseconds, is the time the wave is either (ramping up and) on, or (dropping down and) off. The rise time (**rt**), 10 nanoseconds, is the time given for the wave to ramp from off to on; and the fall time (**ft**), 30 nanoseconds, the time given for the wave to drop from on to off.

```
i4 n4 GND bit ({5(01010 5(1))} pw=10n on=5.0u off=0.0)
```

i4 generates a *repeating* **bit** input. Two distinct patterns are given again, but now *multiplier factors* are included. The wave consists of two alternating patterns: the first pattern contains five bits, the second is a single bit. The five-bit pattern is followed by five successive repetitions of the single-bit pattern, and this combination is repeated five times. (The same pattern could be described by **{5(3(01) 4(1))}**.) The pulse width and on and off voltages are again specified, but the rise and fall times take default values.

```
.vector bb {n5 n6 n7 n8}
ib bb GND bus ({50(Ah) 30(7d4) 20(1000)} pw=5n on=5.0u off=0.0)
```

The **.vector** command defines the bus waveform generated by current source **ib**. The command assigns the bus a name (**bb**) and specifies by name the number of bits the bus waveform will have (four: **n5** through **n8**). The current source statement, which contains the **bus** keyword, specifies waveforms with one or more patterns, along with pulse width and level information.

- The first pattern is **Ah** (hex) = 1010 (binary). Thus, using the names given on the **.vector** command, **n5**=1, **n6**=0, **n7**=1, and **n8**=0. The pattern is repeated 50 times (that is, maintained for a time period equal to the pulse width multiplied by 50).
- The next pattern is **7d4**—that is, 7 (decimal) = 111 (binary), or, to four lower-order bits, 0111. So **n5**=0, **n6**=1, **n7**=1, and **n8**=1. The pattern is repeated 30 times.
- The last pattern is **1000** (binary), so **n5**=1, **n6**=0, **n7**=0, and **n8**=0. The pattern is repeated 20 times.

Current-Controlled Current Source (f)

A two-terminal ideal DC current supply with a level that is a function of one or more control currents.

Syntax

```
fname node1 node2 vname1 K
```

or

```
fname node1 node2 POLY(N) vname1 [vname2 [vname3] P0 P1 P2...
```

name	Current-controlled current source name.
node1	Positive terminal. Positive current flows into ***node1***.
node2	Negative terminal. Positive current flows out of ***node2***.
K	Current gain—the ratio of the output current to the control current.
POLY	Keyword indicating that the output current is a polynomial function of the control currents.
N	Number of control currents.
vname1 vname2 ...	Name(s) of the voltage source(s) supplying the control current(s).
P_0 P_1 P_2 ...	Coefficients of the control polynomial.

Current is reckoned positive if it enters a voltage source at its first terminal. A similar convention holds for the **current-controlled current source**.

The first statement creates a current source with a level equal to ***K*** multiplied by the current through voltage source ***vname1***.

The second statement creates a current source whose level is a nonlinear polynomial function of the currents through up to three voltage sources. Let:

- x = current through voltage source ***vname1***;
- y = current through voltage source ***vname2*** (if ***N*** ≥ 2);
- z = current through voltage source ***vname3*** (if ***N*** ≥ 3).

Then the controlled current source's level is defined as follows:

If ***N*** = 1:

$$P_0 + P_1x + P_2x^2 + P_3x^3 + \ldots \tag{0.15}$$

If ***N*** = 2:

$$P_0 + P_1x + P_2y + P_3x^2 + P_4xy + P_5y^2 + P_6x^3 + P_7x^2y + P_8xy^2 + P_9y^3 + \ldots \tag{0.16}$$

If ***N*** = 3:

$$P_0 + P_1x + P_2y + P_3z + P_4x^2 + P_5xy + P_6xz + P_7y^2 + P_8yz + P_9z^2 + P_{10}x^3 + P_{11}x^2y + P_{12}x^2z + P_{13}xy^2 + P_{14}xyz + P_{15}xz^2 + P_{16}y^3 + P_{17}y^2z + P_{18}yz^2 + P_{19}z^3 + \ldots \tag{0.17}$$

If ***N*** = 1 and only one polynomial coefficient is specified, it is assumed to be P_1, to facilitate the specification of linearly-controlled sources.

Examples

```
ftest in gnd vin 1.0
```

Current-controlled current source **ftest** has a gain of 1 and is controlled by the current through **vin**.

```
f1 0 1 vcntrl 2.0
```

This defines a current source with a level equal to 2 × *i*(**vcntrl**), that is, twice the current through **vcntrl**.

```
f2 0 1 POLY(1) vcntrl 1m 0 2
```

This defines a current source with a level equal to 10^{-3} + (2 × *i*(**vcntrl**)^2).

```
f3 0 1 POLY(2) v1 v2 0 1 2 3
```

This defines a current source with a level equal to *i*(**v1**) + (2 × *i*(**v2**)) + (3 × *i*(**v1**) × *i*(**v2**)).

```
f4 0 1 POLY(3) v1 v2 v3 0 1 0 3 0 4
```

This defines a current source with a level equal to *i*(**v1**) + (3 × *i*(**v3**)) + (4 × *i*(**v1**) × *i*(**v2**)).

Current-Controlled Voltage Source (h)

A two-terminal ideal DC voltage supply with a level that is a function of one or more controlling currents.

Syntax

```
hname node1 node2 vname1 K
```

or

```
hname node1 node2 POLY(N) vname1 [vname2 [vname3 ]] P0 P1 P2 ...
```

name	Current-controlled voltage source name.
node1	Positive terminal.
node2	Negative terminal.
K	Transresistance—the ratio of the output voltage to the control current.
POLY	Keyword indicating that the output voltage is a polynomial function of the control currents.
N	Number of control currents.
vname1 vname2 ...	Name(s) of the voltage source(s) supplying the control current(s).
P_0 P_1 P_2 ...	Coefficients of the control polynomial. Current is reckoned positive if it enters a voltage source at its first terminal.

The first statement creates a voltage source with a level equal to ***K*** multiplied by the current through voltage source ***vname1***.

The second statement creates a voltage source whose level is a nonlinear polynomial function of the currents through up to three voltage sources. Let

- x = current through voltage source ***vname1***;
- y = current through voltage source ***vname2*** (if ***N*** ≥ 2);
- z = current through voltage source ***vname3*** (if ***N*** ≥ 3).

Then the controlled voltage source’s level is defined as follows:

If ***N*** = 1:

$$P_0 + P_1x + P_2x^2 + P_3x^3 + \ldots \tag{0.18}$$

If ***N*** = 2:

$$P_0 + P_1x + P_2y + P_3x^2 + P_4xy + P_5y^2 + P_6x^3 + P_7x^2y + P_8xy^2 + P_9y^3 + \ldots \tag{0.19}$$

If ***N*** = 3:

$$P_0 + P_1x + P_2y + P_3z + P_4x^2 + P_5xy + P_6xz + P_7y^2 + P_8yz + P_9z^2 + P_{10}x^3 + P_{11}x^2y + P_{12}x^2z + P_{13}xy^2 + P_{14}xyz + P_{15}xz^2 + P_{16}y^3 + P_{17}y^2z + P_{18}yz^2 + P_{19}z^3 + \ldots \qquad (0.20)$$

If ***N*** = 1 and only one polynomial coefficient is specified, it is assumed to be *P1*, to facilitate the specification of linearly-controlled sources.

Examples

```
htest in gnd vin 1.23e4
```

Current-controlled voltage source **htest** has a transresistance of 12.3 kilohms and is controlled by the current through **vin**.

```
h1 0 1 vcntrl 2.0
```

This defines a voltage source with a level equal to 2 × *i*(**vcntrl**), that is, twice the current through **vcntrl**.

```
h2 0 1 POLY(1) vcntrl 1m 0 2
```

This defines a voltage source with a level equal to 10^{-3} + (2 × *i*(**vcntrl**)^2).

```
h3 0 1 POLY(2) v1 v2 0 1 2 3
```

This defines a voltage source with a level equal to *i*(**v1**) + (2 × *i*(**v2**)) + (3 × *i*(**v1**) × *i*(**v2**)).

```
h4 0 1 POLY(3) v1 v2 v3 0 1 0 3 0 4
```

This defines a voltage source with a level equal to *i*(**v1**) + (3 × *i*(**v3**)) + (4 × *i*(**v1**) × *i*(**v2**)).

Diode (d)

A two-terminal *p-n* junction diode.

Syntax

```
dname node1 node2 model [[area=]A] [M=M] [tables=T] [L=length] [W=width]
    [PJ=PJ] [LM=LM] [WM=WM] [LP=LP] [WP=WP]
```

name Diode name.

node1 Positive terminal (*p* side).

node2 Negative terminal (*n* side).

model Diode model name. This is specified elsewhere in the input file in the form
.model *name* **d** [*parameters*]
Schottky barrier diodes may be simulated using an appropriate model specification.

A Area of the diode. (Units: unitless for level 1, square meters for level 3. *Default:* 1.)

M Multiplicity—the number of devices to be placed in parallel. (*Default:* 1.)

T Toggle internal tables. When internal tables are on, T-Spice will build a table of current and charge values to speed device evaluation. (*Default:* 0.)

L Length of the diode

W Width of the diode

PJ Junction periphery. Overrides the model PJ value. (Units: Unitless for level 1, meters for level 3.)

LM Length of metal capacitor. Overrides the model LM value. (Units: meters, for level 3 only)

WM Width of metal capacitor. Overrides the model WM value. (Units: meters, for level 3 only)

LP Length of polysilicon capacitor. Overrides the model LP value. (Units: meters, for level 3 only)

WP Width of polysilicon capacitor. Overrides the model WP value. (Units: meters, for level 3 only)

Examples

```
dpn2 n1 n2 dmodel
D3 n3 n4 dmodel 3
```

The **area** factor scales the diode current; thus, **D3** provides three times as much current as **dpn2**, given the same bias conditions.

Inductor (l)

A two-terminal inductor.

Coupled (mutual) inductors can be defined with the **k** statement.

Syntax

```
lname node1 node2 [L=] [M=M] [scale = scale] [tc1 = tc1] [tc2 = tc2] [dtemp
    = dtemp] [r=resistance]
```

or

```
lname node1 node2 L [tc1[tc2]] [M=M] [scale = scale] [dtemp = dtemp]
    [r=resistance]
```

or

```
lname node1 node2 POLY C0 C1 ... [M=M]
```

Parameter	*Symbol*	*Description*
name		Inductor name.
node1		Positive terminal.
node2		Negative terminal.
L	L_0	Inductance. (Unit: henries. Default: 0.)
POLY		Keyword indicating that the inductance is a polynomial function.
C0 C1 ...	c_0, c_1, c_2 ...	Coefficients of the polynomial. The inductance is ***C0*** + (***C1*** × i) + (***C2*** × i^2) ..., where i is the current through the inductor.
M	M	Multiplicity—the number of devices to be placed in parallel. (*Default:* 1.)
scale	S	Element scale factor. (*Default:* 1.)
Tc1	T_{c1}	First temperature coefficient for inductance. (*Unit:* (1/deg C)^2). (*Default:* 0.)
Tc2	T_{c2}	Second temperature coefficient for inductance. (*Unit:* (1/deg C)2). (*Default:* 0.)
Dtemp	D_{temp}	Difference between the inductor and the circuit temperatures. (*Unit:* Deg C). (*Default:* 0.)
R	R_0	Parasitic resistance of the inductor. (*Unit:* ohm). (*Default:* 0.)

The formula for inductance is:

$$L = MS[1 + T_{c1}(\Delta T) + T_{c2}(\Delta T)^2]L_0 \tag{0.21}$$

where

$$\Delta T = T_{circuit} + D_{temp} - T_{nom} \qquad (0.22)$$

where $T_{circuit}$ is set in **.temp** and T_{nom} in **.options *tnom***.

The formula for parasitic resistance is:

$$R_{parasitic} = R_0 / M . \qquad (0.23)$$

Note: When the calculated inductance is greater than or equal to 0.1 H, T-Spice issues a warning message.

Examples

```
L1 na nb 10u
```

The example specifies an inductor with a value of 10 microhenries.

```
L1 a c 25 m=10scale=20R=10 dtemp=20 tc1=1.5e-2 tc2=5e-4
```

Instance (x)

An instantiation of a subcircuit definition or external model device.

Subcircuits can be replicated multiple times to build a hierarchical description of a circuit.

The subcircuit must be defined elsewhere in the input file with a **.subckt/.ends** block.

Nodes several levels deep within a subcircuit hierarchy are named using hierarchical notation in the form **x*instance*.x*instance*.*node***.

The **x** key letter is also used to instance devices that use external user-defined models.

Syntax

```
xname node1 [node2 ...] subcircuit | modelname [parameter=X ...] [M=M]
```

name Subcircuit instance name.

node1 node2 ... Specific instance nodes. The order of nodes named corresponds to the order specified by the **.subckt** command.

parameter=X Parameter(s) from the subcircuit definition or external model whose default value(s) are to be *overridden* by the assignment(s) made here. ***X*** can be a number or an expression. Subcircuit parameters have local scope. Parameters can be written in any order in both definition and instances. Parameters not specified here take their default values.

subcircuit Original subcircuit definition name.

modelname External model name. T-Spice matches this device with a model defined using a **.model** command with a matching modelname and whose type is **external**.

M Multiplicity—the number of representations of parallel instances of the subcircuit. T-Spice multiplies the subcircuit terminal currents and charges by ***M***. ***M*** can be any positive integer or decimal. (*Default:* 1.)

Examples

```
.subckt inv in out Vdd length=1.25u nwidth=2u pwidth=3u
mt1 out in GND GND nmos l='length' w='nwidth'
mt2 out in Vdd Vdd pmos l='length' w='pwidth'
c2 out GND 800f
.ends inv
...
xinv1 a1 a2 Vdd inv nwidth=2.5u
```

The **.subckt/.ends** block creates a three-terminal subcircuit (an inverter) and names it **inv**. The subcircuit consists of two MOSFETs (one *n*-type and one *p*-type) and an 800-femtofarad capacitor. The instance statement defines an instance, named **inv1**, of the inverter subcircuit **inv**. Following the instance name are the three terminals of the instance (in order corresponding to that of the original subcircuit definition); the name of the subcircuit to which it refers; and a new assignment for parameter **nwidth**, which overrides the default value assigned in the definition.

```
x1 1 0 resmodel res=1k
```

This statement instantiates a device with terminals attached to nodes **0** and **1** and matches it to a model defined using

```
.model resmodel external winfile="res.dll"
```

The parameter **res=1k** is passed to the external model.

JFET (j)

A transistor with three terminals: drain, gate, and source. (JFET stands for *junction field effect transistor*.)

Syntax

```
jname drain gate source model [[area=]A] [M=M] [tables=T]
```

name JFET name.

drain Drain terminal.

gate Gate terminal.

source Source terminal.

model JFET model name. This is specified elsewhere in the input file in the form
`.model` *`name`* **`njf|pjf`** [*`parameters`*]

A Area scale factor. (*Default:* 1.)

M Multiplicity—the number of devices to be placed in parallel. (*Default:* 1.)

T Toggle internal tables. When internal tables are on, T-Spice will build a table of current and charge values to speed device evaluation. (*Default:* 0.)

Examples

```
jout 4 8 6 jfet2
j1 vdd in out jfet2 3
```

The **area** factor scales the generated currents; thus, the currents at the terminals of **j1** are three times those at the terminals of **jout**.

MESFET (z)

A transistor with three or four terminals: drain, gate, and source. (MESFET stands for *metal semiconductor field effect transistor*.)

Syntax

z*name drain gate source* [*bulk*] *model* [[**area=**]*A*] [**l=***L*] [**w=***W*] [**M=***M*] [**tables=***T*]

For HSPICE compatibility, you can create a MESFET device in T-Spice using a device name **jname** instead of **zname**. The syntax for these MESFET device statement is the same.]

name MESFET name.

drain Drain terminal.

gate Gate terminal.

source Source terminal.

bulk Bulk terminal.

model MESFET model name. This is specified elsewhere in the input file in the form
.model *name* **nmf** | **pmf** | **njf** | **pjf** [*parameters*]

A Area scale factor. (*Default:* 1.)

L Device length. (*Unit:* meters.)

W Device width. (*Unit:* meters.)

M Multiplicity—the number of devices to be placed in parallel. (*Default:* 1.)

T Toggle internal tables. When internal tables are on, T-Spice will build a table of current and charge values to speed device evaluation. (*Default:* 0.)

Examples

```
zout 4 8 6 mfet2
z1 vdd in out mfet2 3
ztest drain gate source vbg nmes1 w=20u l=2u
```

The **area** factor scales the generated current; thus, the currents at the terminals of **z1** are three times those at the terminals of **zout**. The third example shows specification of the bulk terminal and of width and length. The area is fixed by the given width and length; any **area** specification is overridden by this computed area.

MOSFET (m)

A transistor with four terminals: drain, gate, source, and bulk. (MOSFET stands for *metal oxide semiconductor field effect transistor*.)

Refer to **Additional Model Documentation** for complete documentation of the model parameter variations for each MOSFET level.

Unique T-Spice device model parameters can be found in MOSFET Levels 8, 49 and 53 (BSIM3 Revision 3.3) on page 410, Variables for which equations are not given here are as follows. on page 418, MOSFET Levels 44 and 55 (EKV Revision 2.6) on page 421, and MOSFET Level 57 (BSIM3SOI) on page 422.

Syntax

```
mname drain gate source bulk model [l=L] [w=W] [ad=Ad] [pd=Pd] [as=As]
    [ps=Ps] [nrd=Nrd] [nrs=Nrs] [rdc=Rdc] [rsc=Rsc] [rsh=Rsh] [geo=Geo] [M=M]
    [tables=T]
```

name MOSFET name.

drain Drain terminal.

gate Gate terminal.

source Source terminal.

bulk Bulk terminal.

model MOSFET model name. The model is declared elsewhere in the input file in the form:

```
.model name nmos|pmos
level=1|2|3|4|5|9|13|20|28|30|31|40|47|49|52|100…
[parameters]
```

L Channel length. (*Unit:* meters. *Default:* set by the **.options defl** command.)

W Channel width. (*Unit:* meters. *Default:* set by the **.options defw** command.)

Ad Drain area. (*Unit:* square meters. *Default:* see Drain area on page 166.)

Pd Drain perimeter. (*Unit:* meters. *Default:* see Drain perimeter on page 167.)

As Source area. (*Unit:* square meters. *Default: see* Source area on page 166.)

Ps Source perimeter. (*Unit:* meters. *Default:* see Source perimeter on page 167.)

Nrd Number of squares of diffusion—drain. (*Default:* set by the **.options defnrd** command.)

Nrs Number of squares of diffusion—source. (*Default:* set by the **.options defnrs** command.)

Rdc Additional contact resistance, which overrides the **rdc** model parameter value—drain. (*Unit*: Ohms. *Default*: 0.0)

Rsc Additional contact resistance, which overrides the **rsc** model parameter value—source. (*Unit*: Ohms. *Default*: 0.0)

Rsh Source-Drain sheet resistance, which overrides the rsh model parameter. (*Unit*: Ohms/square. *Default*: 0.0)

Geo Selector for source/drain sharing of stacked devices (*Default*: 0)

M Multiplicity—the number of devices to be placed in parallel. (*Default:* 1.)

T Toggle internal tables. When internal tables are on, T-Spice will build a table of current and charge values to speed device evaluation. (*Default:* 0.)

Default values for ***Ad***, ***Pd***, ***As***, and ***Ps*** depend on the **acm** model parameter. Default value for ***T*** depends on the global modelmode option, which defaults to direct mode (tables=0).

Parasitic diodes are always addes for MOSFETs in direct model evaluation mode. For table based model evaluation, the diodes are replaced with nonlinear capacitors when the option **mosparasitics** is turned off. One diode is placed between the bulk and the source, and another between the bulk and the drain.

The parasitic diode characteristics are determined by the MOSFET device parameters **as**, **ad**, **pd**, **ps**, and **geo**, as well as the MOSFET model parameters **acm**, **cj**, **cjsw**, **cjgate**, **js**, **jsw**, **is**, **n**, **nds**, **vnds**, and **hdif**. The quantity **weff** also plays a role in determining default values for source and drain areas and perimeters for some values of **acm**.

The parasitic diode equations have been modified from the standard diode equations, in an effort to improve compatibility with other SPICE simulators and to improve simulator convergence. The diode charge/capacitance equations are unchanged; they are the same as for regular diodes. The DC current equations for MOSFET parasitic diodes are as follows.

If the MOSFET bulk-source voltage **vbs** is positive (the bulk-source diode is forward-biased), then the bulk-source diode's DC current is given by

$$\mathbf{ibs} = \mathbf{isatbs} \times (\exp(\mathbf{vbs}/(\mathbf{n} \times \mathbf{vt}) - 1)) \tag{0.24}$$

where **vt** = kT/q (the thermal voltage), and the diode's saturation current **isatbs** is

$$\mathbf{isatbs} = (\mathbf{js} \times \mathbf{aseff}) + (\mathbf{jsw} \times \mathbf{pseff}) \tag{0.25}$$

if that value is positive, or is zero otherwise. The effective source area **aseff** and perimeter **pseff** are computed as described below, depending on the value of the **acm** parameter.

Similarly, if the MOSFET bulk-drain voltage **vds** is positive (the bulk-drain diode is forward-biased), the bulk-drain diode's DC current is

$$\mathbf{ibd} = \mathbf{isatbd} \times (\exp(\mathbf{vbd}/(\mathbf{n} \times \mathbf{vt}) - 1)) \tag{0.26}$$

where the diode saturation current **isatbd** is

$$\mathbf{isatbd} = (\mathbf{js} \times \mathbf{adeff}) + (\mathbf{jsw} \times \mathbf{pdeff}) \tag{0.27}$$

if that value is positive, or is otherwise. The effective drain area **adeff** and perimeter **pdeff** are computed as described below.

The exponential function in both diodes is replaced by a linear extension when the current is larger than the value of the **expli** model parameter. The linear extension is chosen such that the diode current function is continuously differentiable at the transition point where the diode current equals **expli**.

The ***n*** parameter is now supported for MOSFET parasitic diodes.

When a MOSFET parasitic diode with saturation current **isat** is reverse-biased with a negative voltage **vdi**, then its current **idi** behaves as follows.

If 0 > **vdi** > **vnds**:

$$\textbf{idi} = \textbf{isat} \times \textbf{vdi} \tag{0.28}$$

If **vdi** < **vnds**:

$$\textbf{idi} = \textbf{isat} \times (\textbf{vnds} + (\textbf{vdi} - \textbf{vnds}) / \textbf{nds}) \tag{0.29}$$

The effective source and drain areas and perimeters are computed as in the table below, depending on the value of the **acm** parameter.

If **acm** = 3, the **geo** device parameter affects these calculations. The **geo** parameter is used to handle stacked MOSFET devices properly, and it can have the following values:

- **geo = 0** (default): the drain and the source are not shared by other devices.
- **geo = 1**: the drain is shared with another device.
- **geo = 2**: the source is shared with another device.
- **geo = 3**: the drain and the source are shared with other devices.

The **geo** parameter may be specified on the MOSFET device statement, at any point after the model name.

Each parasitic diode inherits its multiplicity factor **m** from its "parent" MOSFET. The values of **defas**, **defad**, and **moscap** are specified using **.options**.

	acm = 0	**acm = 1**	**acm = 2**	**acm = 3**
Source area				
with **as**	**as** × **wmlt**^{2}	**weff** × **wmlt**	**as** × **wmlt**^{2}	**as** × **wmlt**^{2}
without **as**	**l** × **w** (if **moscap**=1) **defas** (otherwise)	**weff** × **wmlt**	2 × **hdifeff** × **weff**	2 × **hdifeff** × **weff** (if **geo** = 0 or 1) **hdifeff** × **weff** (otherwise)
Drain area				
with **ad**	**ad** × **wmlt**^{2}	**weff** × **wmlt**	**ad** × **wmlt**^{2}	**ad** × **wmlt**^{2}

	acm = 0	**acm = 1**	**acm = 2**	**acm = 3**
without **ad**	**l** × **w** (if **moscap** = 1) **defad** (otherwise)	**weff** × **wmlt**	2 × **hdifeff** × **weff**	**2** × **hdifeff** × **weff** (if **geo** = 0 or 2) **hdifeff** × **weff** (otherwise)
Source perimeter				
with **ps**	**ps** × **wmlt**	**weff**	**ps** × **wmlt**	**ps** × **wmlt**
without **ps**	2 × (**l**+**w**) (if **moscap** = 1) **defps** (otherwise)	**weff**	4 × **hdifeff** + 2 × **weff**	4 × **hdifeff + weff** (if **geo** = 0 or 1) 2 × **hdifeff** (otherwise)
Drain perimeter				
with **pd**	**pd** × **wmlt**	**weff**	**pd** × **wmlt**	**pd** × **wmlt**
without **pd**	2 × (**l**+**w**) (if **moscap** = 1) **defpd** (otherwise)	**weff**	4 × **hdifeff** + 2 × **weff**	4 × **hdifeff + weff** (if **geo** = 0 or 2) 2 × **hdifeff** (otherwise)

The parasitic drain/source diodes' sidewall capacitance now makes use of the new **cjgate** parameter, which describes the sidewall capacitance per unit length along the gate edge. If **cjgate** is specified and the MOSFET's effective width **weff** is not greater than the diode's perimeter, then the total sidewall capacitance is given by:

$$\mathbf{csw} = \mathbf{cjsw} \times (\boldsymbol{p} - \mathbf{weff}) + (\mathbf{cjgate} \times \mathbf{weff}) \tag{0.30}$$

where ***p*** is **ps** for a source diode or **pd** for a drain diode. Otherwise, if **cjgate** is not specified or **weff** > ***p***, the total sidewall capacitance is:

$$\mathbf{csw} = \mathbf{cjsw} \times \boldsymbol{p} \tag{0.31}$$

Examples

```
m12 n1 n2 GND GND ndep l=10u w=5u ad=100p as=100p pd=40u ps=40u
```

Mutual Inductor (k)

A coupled pair of inductors.

Syntax

```
kname inductor1 inductor2 K
```

name	Mutual inductor name.
inductor1	First inductor.
inductor2	Second inductor.
K	Coefficient of coupling (0 < ***K*** ≤ 1).

Examples

```
k1 La Lb 10u
```

The example illustrates coupling between two inductors **La** and **Lb**, defined elsewhere in the input file:

```
La node1a node2a 10m
Lb node1b node2b 20m
```

The *order* of node naming on the inductor statements determines the relative directions of current flow in the mutual inductor. The current flow from **node1a** to **node2a** (inductor **La**) is in the same direction as from **node1b** to **node2b** (inductor **Lb**). To reverse the current flow in either inductor, reverse the node order on the appropriate inductor statement.

Resistor (r)

A two-terminal resistor.

The resistance *R* is influenced by the temperature as follows:

$$R = \mathbf{N}(1 + \mathbf{A}T + \mathbf{B}T^2)$$
$$T = Ta - Tn$$

where ***N***, ***A***, ***B*** are device parameters described below; *Ta* (the "ambient" temperature) is set by the **.temp** command; and *Tn* (the "nominal" temperature) is set by the **.options tnom** command.

Resistors can be specified using geometric and physical parameters such as ***l***, ***w***, and ***rsh***. For a description of how resistance is calculated using these parameters, refer to the device model Resistor on page 440.

Optional capacitors may be included between the terminals and a bulk node (usually ground) to obtain a simple transmission line model.

Syntax

```
rname node1 node2 r=r [resistor_parameters]
```

or

```
rname node1 node2 r [tc1[tc2]] [resistor_parameters]
```

or

```
rname node1 node2 modelname + [[r=r] [resistor_parameters]
```

or

```
rname node1 node2 modelname r [tc1[tc2]] [resistor_parameters]
```

name	Resistor name.
node1	Positive terminal.
node2	Negative terminal.
modelname	Name of resistor model. Must match **.model *name*** when ***type*** is **r**. For additional information, see **.model** (page 85).
r=*resistance*	Nominal resistance. (*Unit:* ohms.)

In the first syntax and the third syntax, the ***resistor_parameters*** field is of the form:

```
[tc1=tc1] [tc2=tc2] [noise=noise] [m=mult] [scale=devscale] [ac=acres]
    [dtemp=dtemp] [l=l] [w=w] [c=c]
```

In the second syntax and the fourth syntax, the ***resistor_parameters*** field is of the form:

```
[noise=noise] [m=mult] [scale=scale] [ac=acres] [dtemp=dtemp] [l=l] [w=w]
   [c=c]
```

Parameter	*Description*
tc1	First-order temperature coefficient. (Default: resistor model parameter ***tc1r***; 0 if no model is specified.)
tc2	Second-order temperature coefficient. (Default: resistor model parameter ***tc2r***; 0 if no model is specified.)
noise	Noise source multiplier. **noise=0** eliminates resistor noise. (Default: resistor model parameter ***noise***; 1 if no model is specified.)
mult	Multiplicity—the number of devices to be placed in parallel. (*Default:* 1.)
devscale	Multiplies resistance and capacitance of device. (*Default:* 1.)
acres	Specifies device resistance during AC analysis. (*Default:* resistor model parameter ***rac***; if no model is specified, default is DC resistance.)
dtemp	Specifies the difference between the device temperature and the general circuit operating temperature. (*Default:* 0.)
l	Resistor length. Scaled length is obtained by multiplying ***l*** by **.options scale** (not the element parameter ***devscale***, above) or the resistor model parameter ***shrink***. (*Default:* resistor model parameter ***l***.)
w	Resistor width. Scaled width is obtained by multiplying ***w*** by **.options scale** (not the element parameter ***devscale***, above) or the resistor model parameter ***shrink***. (*Default:* resistor model parameter ***w***.)
c	Capacitance between node2 and a bulk node specified as a model parameter. Multiplied by **.options scale**. (*Default:* resistor model parameter ***cap***.)

Note: If T-Spice calculates the effective resistance, R_{eff}, to be less than 10^{-5} Ω, then a warning message is issued and the effective resistance is automatically assigned a value of 10^{-5} Ω. See the model description Resistor on page 440 for calculation of R_{eff}.

Examples

```
r1 2 1 30K TC=1e-2,1e-4
```

This produces a resistor of resistance 30 kilohms at the nominal temperature tnom. If the temperature T is different from ***tnom***, the resistance is 30,000*(1+0.01*(T-tnom)+0.0001*(T-tnom)*(T-tnom)). For example, if the circuit temperature is 127 degrees and ***tnom*** is 27 degrees, the resistance is 30,000*(1+0.01*100+0.0001*100*100) = 90,000 Ohms.

```
r1 n1 n2 rmod l=5u w=10u
.model rmod r rsh=1k cap=10pf cratio=0.5
```

This example creates a resistor between nodes **n1** and **n2** of size 500 Ohms (***r***= ***l****rsh*/***w***) as well as two capacitors, 5 picofarads each, one between **n1** and **Gnd**, the other between ***n2*** and **Gnd.**

Voltage- or Current-Controlled Switch (s)

A switch is implemented as a resistor between ***node1*** and ***node2***, whose resistance is controlled by the controlling voltage or current. (See also the device model Switch on page 443.)

Syntax

The general syntax for T-Spice's voltage-controlled switch element is:

```
sname node1 node2 control1 control2 modelname
```

The syntax for a current-controlled switch is:

```
sname node1 node2 vsourcename modelname
```

name	Switch name.
node1	Positive terminal.
node2	Negative terminal.
control1	Name of the voltage source supplying the control voltage.
control2	Name of the voltage source supplying the control voltage.
vsourcename	Controlling current for a current-controlled switch.
modelname	Name of resistor model. Must match **.model** ***modelname*** when ***type*** is **sw** or **csw**. For additional information, see **.model** (page 85).

Resistance is ***roff*** when the switch is off and ***ron*** when the switch is on. The switch is on when the control voltage or current for the switch is greater than its threshold voltage or current.

The ***ron***, ***roff***, and threshold values for the switch are specified in a **.model** statement whose model name matches the ***modelname*** on the device statement.

T-Spice switch elements can display hysteresis, so that the threshold value is different when the control voltage/current is increasing than when it is decreasing. For a voltage-controlled switch, the threshold voltage is ***vt*** when **v(*control1, control2*)** is increasing, and ***vt-vh*** when **v(*control1, control2*)** is decreasing. For a current-controlled switch the threshold current is ***it*** when **i(*vsourcename*)** is increasing, and ***it-ih*** when **i(*vsourcename*)** is decreasing. The switch is on when the control voltage or current is greater than the threshold value.

The ***dv*** and ***di*** parameters define a small interval around the threshold in which a smooth transition between ***ron*** and ***roff*** is made.

Examples

Voltage-Controlled Switch

The following two examples would both produce voltage-controlled switches:

```
s1 n1 n2 c1 c2 swmod
.model swmod sw ron=1 roff=1e12 vt=2.5
```

and

```
s1 out 0 0 in swmod
.model swmod sw vt=0 dv=0.2
```

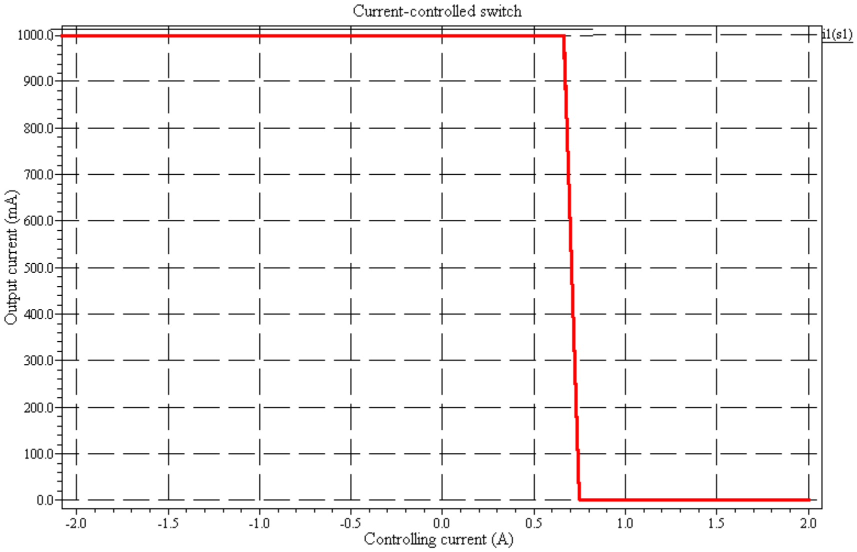
The waveform for this switch is illustrated in Voltage-Controlled Resistor on page 183.

Current-Controlled Switch

The following example demonstrates the modeling of a current-controlled switch:

```
s1 out 0 vin swmod
.model swmod csw it=0.7 di=0.1
```

This switch would produce the following waveform:



Transmission Line (t)

A mechanism for “lossless” or “lossy” signal propagation.

The “lossless” transmission line is described by characteristic impedance and delay; the “lossy” transmission line is described by RLCG parameters.

Syntax

“Lossless” Transmission Line

```
tname node1 node2 node3 node4 z0=Z [td=D] [f=F [nl=N]]
```

name Transmission line name.

node1 ... ***node4*** Terminals. ***node1*** (+) and ***node2*** (–) are at one end of the transmission line, ***node3*** (+) and ***node4*** (–) at the opposite end.

Z Impedance. (*Unit:* ohms.)

D Transmission delay. The delay may instead be specified indirectly from ***F*** and ***N***. (*Unit:* seconds.)

F Line frequency. (*Unit:* Hertz. *Default:* 1 × 10^9.)

N Normalized number of wavelengths. The transmission delay is the ratio of the wavelength number ***N*** to the line frequency ***F***. (*Default:* 0.25.)

“Lossy” Transmission Line

```
tname node1 node2 node3 node4 r=R l=L c=C g=G length=W[lumps=X][lumptype=Y]
```

name Transmission line name.

node1 ... ***node4*** Terminals. ***node1*** (+) and ***node2*** (–) are at one end of the transmission line, ***node3*** (+) and ***node4*** (–) at the opposite end.

R Distributed resistance. (*Unit:* ohms ⁄ meter.)

L Distributed inductance. (*Unit:* henries ⁄ meter.)

C Distributed capacitance. (*Unit:* farads ⁄ meter.)

G Distributed conductance. (*Unit:* siemens ⁄ meter.)

W Physical length. (*Unit:* meters.)

X Number of lumps used for iterative ladder circuit (ILC) expansion. (*Default:* 1.)

Y Type of lumps used for ILC expansion (see below). (*Default:* 3.)

Y is one of the following:

0	“Gamma” type lumps.
1	“Tee” type symmetric lumps.
2	“Pi” type symmetric lumps.
3	Hybrid RGT lumps (default).

Examples

```
tline2 pad2 GND pin2 GND z0=100 td=10ns
tline3 drive GND out GND z0=300 f=100meg nl=.1
```

Voltage Source (v)

A two-terminal ideal voltage supply.

Exponential, pulse, piecewise linear, frequency-modulated, sinusoidal, and customizable (vectorized) waveforms are available.

Voltage sources whose waveform is described using an expression can be created using the **e**-element with an expression and the **time()** function.

Syntax

```
vname node1 node2 [[DC] V] [AC M [P]] [waveform]
```

name	Voltage source name.
node1	Positive terminal—or bus named by an associated **.vector** command.
node2	Negative terminal.
V	DC level between ***node1*** and ***node2***. (*Unit:* volts. *Default:* 0.)
waveform	Waveform identifier and parameters (see below).
M	AC magnitude. (*Unit:* volts.)
P	AC phase. (*Unit:* degrees. *Default:* 0.)

DC, AC, and transient values can be specified independently and in any order.

waveform is one of the following:

Exponential Waveform

```
exp (Vi Vp [Dr [Tr [Df [Tf]]]])
```

Vi	Initial voltage. (*Unit:* volts.)
Vp	Peak voltage. (*Unit:* volts.)
Dr	Rise time delay. (*Unit:* seconds. *Default:* 0.)
Tr	Rise time constant. (*Unit:* seconds. *Default:* 0.)
Df	Fall time delay. (*Unit:* seconds. *Default:* 0.)
Tf	Fall time constant. (*Unit:* seconds. *Default:* 0.)

Pulse Waveform

```
pulse (Vi Vp [D [Tr [Tf [Pw [Pp]]]]]) [ROUND=RND]
```

Vi	Initial voltage. (*Unit:* volts.)

Vp Peak voltage. (*Unit:* volts.)

D Initial delay. (*Unit:* seconds. *Default:* 0.)

Tr Rise time. (*Unit:* seconds. *Default:* time step from **.tran**.)

Tf Fall time. (*Unit:* seconds. *Default:* time step from **.tran**.)

Pw Pulse width. (*Unit:* seconds. *Default:* stop time from **.tran**.)

Pp Pulse period. (*Unit:* seconds. *Default:* stop time from **.tran**.)

RND Rounding half-interval. A corner at time *T* is replaced by a smoothly differentiable polynomial in the interval (*T*–***RND***,*T*+***RND***). The maximum ***RND*** is half the distance to the nearest neighboring corner. (*Default:* 0—no rounding.)

Note that rise time is not necessarily a 'rise' time, but is the time to go from the initial voltage to the pulse voltage, regardless of whether it's smaller or larger.

Piecewise Linear Waveform

```
pwl (T1 V1 [T2 V2 ...])[ROUND=RND] [REPEAT[=Tr]] [TD=DELAY]
```

T1 T2 Time at corner 1, 2, and so on. (*Unit:* seconds.)

V1 V2 Voltage at corner 1, 2, and so on. (*Unit:* volts.)

ROUND Rounding half-interval. A corner at time *T* is replaced by a smoothly differentiable polynomial in the interval (*T*–***RND***,*T*+***RND***). The maximum ***RND*** is half the distance to the nearest neighboring corner. (*Default:* 0—no rounding.)

REPEAT Starting time within the specified waveform for an infinite number of repetitions of the subwaveform. If ***Tr*** is not specified, the entire waveform repeats indefinitely (i.e., ***Tr***=0). ***Tr*** must be less than or equal to the duration of the waveform. Waveforms can only repeat if the start and end points match. If they do not match, the repeat option is ignored. The **REPEAT** keyword can be abbreviated to **R**.

TD Time delay added to the beginning of the waveform. If you specify corners ***T1, T2,*** *etc*. and **TD**=***DELAY***, then the defined voltage values will actually be applied at effective corner times ***T1***+***DELAY,*** ***T2***+***DELAY,*** *etc*.

Piecewise Linear Waveform File

```
pwlfile filename [ROUND=RND] [REPEAT[=Tr]] [TD=DELAY]
```

filename Input file which contains the piecewise linear waveform definition in a series of **time, voltage** pairs, one per line.

ROUND Same meaning as with **pwl** waveforms

REPEAT Same meaning as with **pwl** waveforms

TD Same meaning as with **pwl** waveforms

Frequency-Modulated Waveform

```
sffm (Vo Vp [Fc [Xm [Fs]]])
```

Vo Offset voltage. (*Unit:* volts.)

Vp Peak voltage. (*Unit:* volts.)

Fc Carrier frequency (*Unit:* Hertz. *Default:* $1/T$, where T is the stop time from **.tran**.)

Xm Modulation index. (*Default:* 0.)

Fs Signal frequency. (*Unit:* Hertz. *Default:* $1/T$, where T is the stop time from **.tran**.)

Sinusoidal Waveform

```
sin (Vo Vp [Fr [De [Da [Ph]]]])
```

Vo Offset voltage. (*Unit:* volts.)

Vp Peak voltage. (*Unit:* volts.)

Fr Frequency. (*Unit:* Hertz. *Default:* $1/T$, where T is the stop time from **.tran**.)

De Delay time. (*Unit:* seconds.)

Da Damping factor. (*Unit:* 1/seconds.)

Ph Phase advance. (*Unit:* degrees.)

Vectorized Waveform

```
bit|bus ({pattern} [on=A] [off=Z] [delay=D] [pw=P] [rt=R] [ft=F] [lt=L]
    [ht=H]) [ROUND=RND]
```

pattern An expression consisting of one or more string or string-multiplier combinations (see below).

A On voltage (*Unit:* volts. *Default:* 0.001.)

Z Off voltage (*Unit:* volts. *Default:* 0.)

D Delay time. (*Unit:* seconds. *Default:* 0.)

P Pulse width: $P = R + T_A = F + T_Z$, where T_A is the time during a pulse where the voltage is "on" ($V = A$) and TZ is the time during a pulse where the voltage is "off" ($V = Z$). (*Unit:* seconds. *Default:* 10×10^{-9}.)

R Rise time. (*Unit:* seconds. *Default:* 1×10^{-9}.)

F Fall time. (*Unit:* seconds. *Default:* 1×10^{-9}.)

L Low time: $L = F + T_Z$, where T_Z is the time during a pulse where the voltage is "off" ($V = Z$). (*Unit:* seconds. *Default:* 10×10^{-9}.)

H	High time: ***H*** = ***R*** + T_A, where T_A is the time during a pulse where the voltage is "on" (*V* = ***A***). (*Unit:* seconds. *Default:* 10 × 10^{-9}.)
RND	Rounding half-interval. A corner at time *T* is replaced by a smoothly differentiable polynomial in the interval (*T*–***RND***,*T*+***RND***). The maximum ***RND*** is half the distance to the nearest neighboring corner. (*Default:* 0—no rounding.)

A *bit* pattern consists of a set of numbers (possibly associated with multiplier factors) whose binary representations sequentially specify the "on"/"off" structure of the waveform. The pattern takes the form ***a*(*b*(*x*) *c*(*y*) …)**, where ***a***, ***b***, and ***c*** are the optional multiplier factors and ***x*** and ***y*** are the numbers.

A *bus* pattern consists of a set of numbers (possibly associated with multiplier factors) whose binary representations—"bit strings"—are grouped together as a waveform bus and treated as a single input. The length of the bit strings is specified by the **.vector** command. If there are *n* nodes in a vector, then T-Spice assigns the first *n* bits of each bit string to those nodes. Extra bits are discarded. If there are not enough bits, the highest-order bits are set to zero. The leftmost node name in the **.vector** command takes the most significant bit.

Numbers are specified on the device statement in binary, hexadecimal (suffixed by **h**), octal (suffixed by **o**), or decimal (suffixed by **d**) notation. (For decimal representations the number of lower-order bits to be collected is also given.)

Examples

```
v1 n1 GND sin (2.5 2.5 30MEG 100n)
```

v1 generates a **sin** (sinusoidal) input. It has an amplitude of 2.5 volts, a frequency of 30 MHz, an offset of 2.5 volts from system ground, and a time delay of 100 nanoseconds after the start of the simulation before the wave begins.

```
v2 n2 GND bit ({01010 11011} on=5.0 off=0.0 pw=50n rt=10n ft=30n)
```

v2 generates a **bit** input. Enclosed in braces **{ }** are two binary-valued five-bit patterns specifying the waveform. The two patterns alternate in time. The **on** voltage value is 5.0 volts; the **off** voltage value is zero. The pulse width (**pw**), 50 nanoseconds, is the time the wave is either (ramping up and) on, or (dropping down and) off. The rise time (**rt**), 10 nanoseconds, is the time given for the wave to ramp from off to on; and the fall time (**ft**), 30 nanoseconds, the time given for the wave to drop from on to off.

```
v3 n3 GND bit ({5(01010 5(1))} pw=10n on=5.0 off=0.0)
```

v3 generates a *repeating* **bit** input. Two distinct patterns are given again, but now *multiplier factors* are included. The wave consists of two alternating patterns: the first pattern contains five bits, the second is a single bit. The five-bit pattern is followed by five successive repetitions of the single-bit pattern, and this combination is repeated five times. (The same pattern could be described by **{5(3(01) 4(1))}**.) The pulse width and on and off voltages are again specified, but the rise and fall times take default values.

```
.vector bb {n7 n6 n5 n4}
vb bb GND bus ({50(Ah) 30(7d4) 20(1000)} pw=5n on=5.0 off=0.0)
```

The **.vector** command defines the bus waveform generated by voltage source **vb**. The command assigns the bus a name (**bb**) and specifies by name the number of bits the bus waveform will have (four: **n7** through **n4**). The voltage source statement, which contains the **bus** keyword, specifies waveforms with one or more patterns, along with pulse width and level information.

- The first pattern is **Ah** (hex) = 1010 (binary). Thus, using the names given on the **.vector** command, **n7**=1, **n6**=0, **n5**=1, and **n4**=0. The pattern is repeated 50 times (that is, maintained for a time period equal to the pulse width multiplied by 50).
- The next pattern is **7d4**—that is, 7 (decimal) = 111 (binary), or, to four lower-order bits, 0111. So **n7**=0, **n6**=1, **n5**=1, and **n4**=1. The pattern is repeated 30 times.
- The last pattern is **1000** (binary), so **n74**=1, **n6**=0, **n5**=0, and **n4**=0. The pattern is repeated 20 times.

Voltage-Controlled Current Source (g)

A two-terminal ideal DC current supply with a level that is a function of one or more controlling voltages. This device can be utilized to model a wide variety of elements, including voltage-controlled resistors, nonlinear capacitors, voltage-controlled capacitors, switch-level MOSFETs, and diodes. See Examples on page 183.

Syntax

g*name node1 node2 na1 nb1 K*

or

g*name node1 node2* **POLY(***N***)** *na1 nb2* [*na2 mb2* ...] P_0 P_1 P_2 ...

or

g*name node1 node2* **LAPLACE** *na1 nb1* a_1 a_2 ... a_m [/ b_1 b_2 ... b_m]

or

g*name node1 node2* [**cur**='*expression*'] [**chg**='*expression*']

name	Voltage-controlled current source name.
node1	Positive terminal. Positive current flows into ***node1***.
node2	Negative terminal. Positive current flows out of ***node2***.
K	Transconductance—the ratio of the output current to the control voltage.
POLY	Keyword indicating that the output current is a polynomial function of the control voltages.
N	Number of control voltages.
LAPLACE	Keyword indicating that the output current is described via a Laplace transform function
cur	Keyword indicating an expression ***expression*** that defines current flowing through the device.
chg	Keyword indicating an expression ***expression*** that defines terminal charges of the device. Used to define the capacitance of a nonlinear capacitor.
expression	Expression involving any node voltages and source currents.
na1 nb2...	Node pairs whose voltages control the level of the **g** element.
P_0 P_1 P_2 ...	Coefficients of the control polynomial.
a_0 a_1 a_2 ... a_m	Numerator of Laplace transfer function.
b_0 b_1 b_2 ...a_n	Denominator of Laplace transfer function.

Current is reckoned positive if it enters the **g** element at its first terminal.

Linear Functions

The first form of voltage-controlled current sources creates a current source with a level equal to ***K*** multiplied by the voltage across the node pair ***na1 nb1***.

Polynomial Functions

The second form creates a current source whose level is a nonlinear polynomial function of the voltages across one or more node pairs. Let

- x = voltage across node pair ***na1 nb1***;
- y = voltage across node pair ***na2 nb2*** (if ***N*** ≥ 2);
- z = voltage across node pair ***na3 nb3*** (if ***N*** ≥ 3).

Then the controlled voltage source's level is defined as follows:

If ***N*** = 1:

$$P_0 + P_1 x + P_2 x^2 + p_3 x^3 + \ldots \tag{0.32}$$

If ***N*** = 2:

$$P_0 + P_1 x + P_2 y + P_3 x^2 + P_4 xy + P_5 y^2 + P_6 x^3 + P_7 x^2 y + P_8 xy^2 + P_9 y^3 + \ldots \tag{0.33}$$

If ***N*** = 3:

$$P_0 + P_1 x + P_2 y + P_3 z + P_4 x^2 + P_5 xy + P_6 xz + P_7 y^2 + P_8 yz + P_9 z^2 + P_{10} x^3 + P_{11} x^2 y + P_{12} x^2 z + P_{13} xy^2 + P_{14} xyz + P_{15} xz^2 + P_{16} y^3 + P_{17} y^2 z + P_{18} yz^2 + P_{19} z^3 + \ldots \tag{0.34}$$

If ***N*** = 1 and only one polynomial coefficient is specified, it is assumed to be *P1*, to facilitate the specification of linearly-controlled sources.

Laplace Functions

With the Laplace keyword, the current source is implemented as a Laplace transfer function.

$$H(s) = \frac{a_0 + a_1 s + a_2 s^2 + \ldots + a_m s^m}{b_0 + b_1 s + b_2 s^2 + \ldots + b_n s^n} \tag{0.35}$$

Expression-Controlled Functions

The fourth form of voltage controlled current sources uses mathematical expressions (**Expressions** (page 46)) to define the output current and charge functions. At least one of the keywords **cur** or **chg** must be specified.

Examples

```
Gtest  in     out    n10    n17    -2.314
```

Gtest is a **g** element. Its terminals are connected to nodes **in** and **out**. The voltage across the node pair **n10** and **n17** control the level of **gtest**. The level of **gtest** equals *(-2.314){v(n10) – v(n17)}*. If this is a positive number, current flows in this direction: node **in**-**gtest** -node **out**.

```
G1     0      1      poly(1)
+                          n10    n17
+                                        1m     0      2
```

g1 is a **g** element. It terminals are connected to nodes **0** and **1**. The level of **g1** is a polynomial in one variable. The one variable is the voltage across the node pair **n10** and **n17**. The level of **g1** is computed as

$$g1 = 10^{-3} + 2\{v(n10) - v(n17)\}^2 \tag{0.36}$$

If this is a positive number, current flows in this direction: node **0**–**g1**–node **1**.

```
G3     0      1      poly(3)
+                          nking  nkong
+                          nping  npong
+                          nsing  nsong
+                                        0      1      0      304
```

g3 is a **g** element. Its terminals are connected to nodes **0** and **1**. The level of **g3** is a polynomial in three variables. The three variables are the voltages across the three node pairs **nking** and **nkong**, **nping** and **npong**, **nsing** and **nsong**. The level of **g3** is computed as

$$\begin{aligned} g3 = {} & \{v(nking) - v(nkong)\} + 3\{v(nsing) - v(nsong)\} \\ & + 4\{v(nking) - v(nkong)\}\{v(nping) - v(npong)\} \end{aligned} \tag{0.37}$$

If this is a positive number, current flows in this direction: node **0**–**g3**–node **3**.

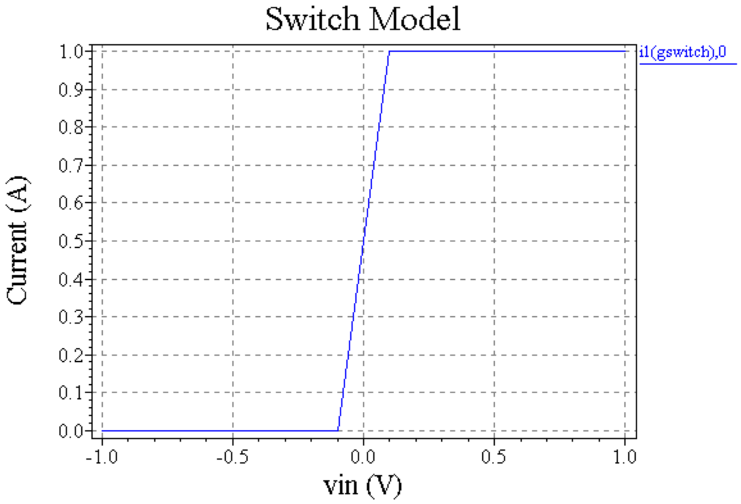
Laplace Transforms

For examples of Laplace transforms, please refer to the analogous examples of Laplace transforms in the voltage-controlled voltage source section (**Voltage-Controlled Voltage Source (e)** (page 187)).

Voltage-Controlled Resistor

```
gswitch out 0 cur='v(out)*table(v(in), -1,1e-12, -0.1,1e-12, 0.1,1, 1,1)'
```

The switch's resistance between nodes **out** and **ground** is controlled by the voltage at node **in**. The switch is off when **v(in**) is less than -0.1 V, and on when **v(in)** is greater than 0.1 V. The interval -0.1V<v(in)<0.1V serves as a smooth transition between the on and off states. Note the use of the table function to describe the conductance characteristics of the switch: the conductance is 10^{-12} (corresponding to a resistance of 10^{+12}) when the switch is off, while the conductance is 1 when the switch is on. The chart below shows the switch's output current as a function of input voltage (holding the voltage at node out fixed at 1V).



Nonlinear Capacitor

The following example models a nonlinear capacitor whose capacitance is a function of applied voltage. A capacitor is described by specifying a charge function that depends on the voltage across the device. The device's capacitance is the derivative of the charge function with respect to the voltage. For example, a CMOS capacitor might be modeled as follows:

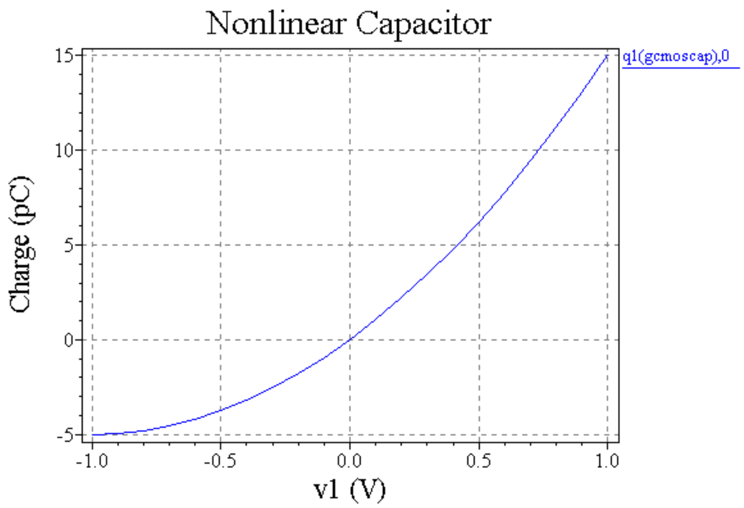
```
.param c0=10p vcc=1
gcmoscap n+ n- chg='c0*v(n+,n-) * (1 + 0.5*vcc*v(n+,n-))'
```

The capacitance of the device **gcmoscap** is then given by:

$$C = c0 * (1 + vcc * v(n+,n-)) \qquad (0.38)$$

where *c0* represents the capacitance at zero applied voltage, and *vcc* measures the sensitivity of the capacitance with respect to input voltage.

The capacitor's charge depends on applied voltage, as shown in the chart below.



Voltage-Controlled Capacitor

A voltage-controlled capacitor is a two-terminal device whose capacitance is a function of node voltages elsewhere in the circuit. Such an element can be modeled in T-Spice with the expression-controlled g-element, using the **chg** parameter.

For example, the capacitance of a vertically moving parallel plate capacitor, a device used in the design of microelectromechanical systems (MEMS) might be modeled as follows:

```
gvccap n+ n- chg='v(n+,n-)*k/v(gapdistance)'
```

where ***gapdistance*** refers to a state variable which represents the distance between the capacitor's plates and ***k*** is a proportionality constant defined using **.param** (page 102).

Switch-Level MOSFET

The T-Spice **stp()** and **table()** functions can be used to create a switch-level model for a MOSFET. An example of such a model for an N-type MOSFET is as follows:

```
gmos d s cur='v(d,s) * table(v(g,s)*stp(v(d,s)) + v(g,d)*(1-stp(v(d,s))),
+                      0,1e-12, 0.4,1e-12, 1,1e-7, 2,2e-5,
+                      3, 1e-4, 5, 4e-4)'
```

Note that the use of the **stp()** function allows for a model that is symmetric with respect to source and drain.

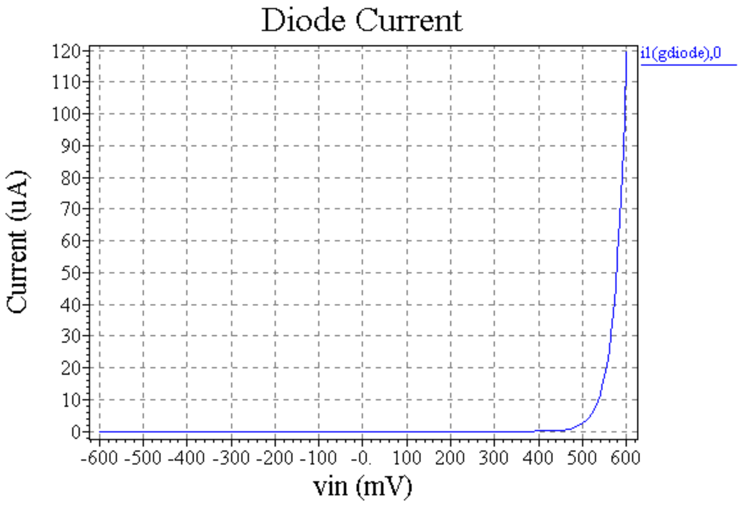
Diode

The T-Spice **g**-element can be used to model any device for which analytic equations are available. When the equations have different forms for different regions of operation, the T-Spice **stp()** function can be useful. The following example models the current and capacitance of a diode. Note that the capacitance equation has different forms for the forward and reverse bias regions, but the **stp()** function allows us to describe the entire model using a single charge expression.

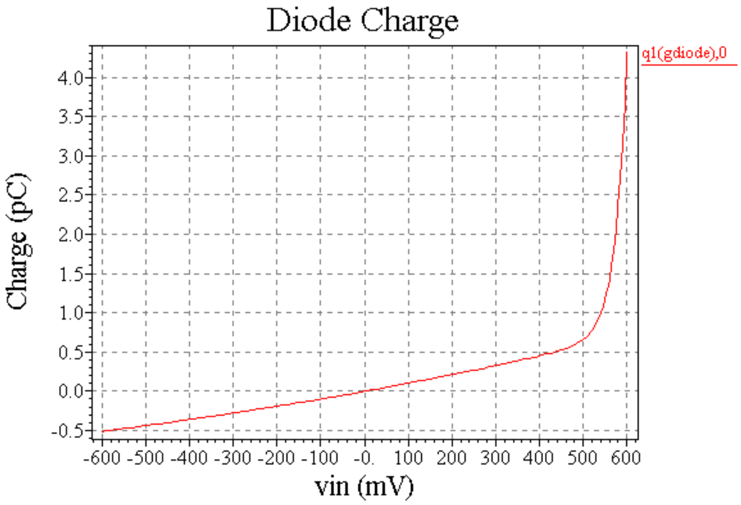
```
.param vt=0.02586 is=1e-14 tt=30n cjo=1e-12 vj=1 m=0.5
```

```
gdiode 1 2 cur='is*(exp(v(1,2)/vt)-1)'
+          chg='tt*is*(exp(v(1,2)/vt)-1)
+               + cjo*(2*vj*(1-sqrt(1-v(1,2)/vj))*stp(v(1,2)))
```

This example would produce the following waveform for current:



and the following waveform for charge:



Voltage-Controlled Voltage Source (e)

A two-terminal ideal DC voltage supply with a level that is a function of one or more controlling voltages.

Syntax

e*name node1 node2 na1 nb1 K*

or

e*name node1 node2* **POLY(***N***)** *na1 nb1* [*na2 nb2*...] P_0 P_1 P_2 ..

or

e*name node1 node2* **LAPLACE** *na1 nb1* a_1 a_2 ... a_m [/ b_1 b_2 ... b_n]

or

e*name node1 node2* **vol**='*expression*'

name Voltage-controlled voltage source name.

node1 Positive terminal.

node2 Negative terminal.

K Voltage gain—the ratio of the output voltage to the control voltage.

POLY Keyword indicating that the output voltage is a polynomial function of the control voltages.

N Number of control voltages.

LAPLACE Keyword indicating that the output current is described via a Laplace transform function

vol Keyword indicating that the source voltage is specified by an expression.

expression Expression specified by the **vol** keyword.

na1 nb2... Node pairs whose voltages control the level of the **e** element.

P_0 P_1 P_2 ... Coefficients of the control polynomial.

a_0 a_1 a_2 ... a_m Numerator of Laplace transfer function.

b_0 b_1 b_2 ...a_n Denominator of Laplace transfer function.

Linear Functions

The first statement creates a voltage source with a level equal to ***K*** multiplied by the voltage across node pair ***na1 nb1***.

Polynomial Functions

The second statement creates a voltage source whose level is a nonlinear polynomial function of the voltages supplied by up to three voltage sources. Let

- x = voltage across node pair ***na1 nb1***;
- y = voltage across node pair ***na2 nb2*** (if ***N*** ≥ 2);
- z = voltage across node pair ***na3 nb3*** (if ***N*** ≥ 3).

Then the controlled voltage source's level is defined as follows:

If ***N*** = 1:

$$P_0 + P_1x + P_2x^2 + P_3x^3 + \ldots \tag{0.39}$$

If ***N*** = 2:

$$P_0 + P_1x + P_2y + P_3x^2 + P_4xy + P_5y^2 + P_6x^3 + P_7x^2y + P_8xy^2 + P_9y^3 + \ldots \tag{0.40}$$

If ***N*** = 3:

$$P_0 + P_1x + P_2y + P_3z + P_4x^2 + P_5xy + P_6xz + P_7y^2 + P_8yz + P_9z^2 + P_{10}x^3 + P_{11}x^2y + P_{12}x^2z + P_{13}xy^2 + P_{14}xyz + P_{15}xz^2 + P_{16}y^3 + P_{17}y^2z + P_{18}yz^2 + P_{19}z^3 + \ldots \tag{0.41}$$

If ***N*** = 1 and only one polynomial coefficient is specified, it is assumed to be *P1*, to facilitate the specification of linearly-controlled sources.

Laplace Functions

With the Laplace keyword, the current source is implemented as a Laplace transfer function.

$$H(s) = \frac{a_0 + a_1s + a_2s^2 + \ldots + a_ms^m}{b_0 + b_1s + b_2s^2 + \ldots + b_ns^n} \tag{0.42}$$

Expression-Controlled Functions

The fourth form of voltage controlled current sources uses mathematical expressions (**Expressions** (page 46)) to define the output current and charge functions. At least one of the keywords **cur** or **chg** must be specified.

Examples

```
Etest  in     out    n10    n17    -2.314
```

Etest is an **e** element. Its nodes are connected to nodes **in** and **out**. The voltage across the node pair **n10** and **n17** control the level of **etest**. The level of **etest** equals *(-2.314){v(n10) – v(n17)}*.

```
E1      0      1      poly(1)
+                           n10    n17
+                                        1m     0      2
```

e1 is an **e** element. It terminals are connected to nodes **0** and **1**. The level of **e1** is a polynomial in one variable. The one variable is the voltage across the node pair **n10** and **n17**. The level of **e1** is computed as

$$e1 = 10^{-3} + 2\{v(n10) - v(n17)\}^2 \tag{0.43}$$

```
E3      0      1      poly(3)
+                           nking  nkong
+                           nping  npong
+                           nsing  nsong
+                                        0      1      0      304
```

e3 is an **e** element. Its terminals are connected to nodes **0** and **1**. The level of **e3** is a polynomial in three variables. The three variables are the voltages across the three node pairs **nking** and **nkong**, **nping** and **npong**, **nsing** and **nsong**. The level of **e3** is computed as

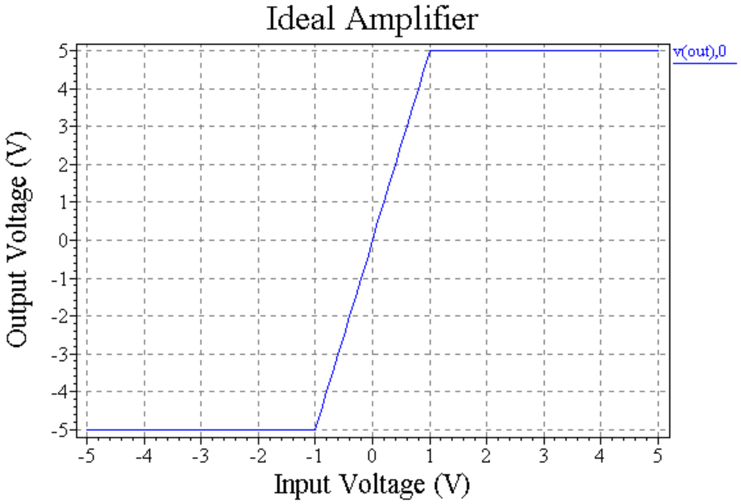
$$\begin{aligned} e3 = {} & \{v(nking) - v(nkong)\} + 3\{v(nsing) - v(nsong)\} \\ & + 4\{v(nking) - v(nkong)\}\{v(nping) - v(npong)\} \end{aligned} \tag{0.44}$$

Ideal OpAmp

The expression-controlled e-element (voltage-controlled voltage source) can be used with the table function to model an ideal voltage amplifier. The following circuit element implements a voltage amplifier with a gain of 5, and minimum and maximum output voltages of -5V and 5V, respectively.

```
eamp out 0 vol='table(v(in), -1, -5, 1, 5)'
```

This example would produce the following waveform:



Integrator element

You can model an integrator using the e element with the Laplace transformation function.

Consider the behavior of an integrator. In the frequency domain, the integrator is modeled as:

$$V_{out} = \frac{k}{s}V_{in} \tag{0.45}$$

And, in the time domain the integrator is modeled as:

$$V_{out} = k\int V_{in}dt \tag{0.46}$$

The transfer function for the voltage gain is:

$$H(s) = \frac{V_{out}}{V_{in}} = \frac{k}{s} \tag{0.47}$$

This is equivalent to the Laplace transfer function with the coefficient assignments:
$a_0 = k, a_1=0, b_0=0, b_1=1$

The voltage-controlled voltage source element which implements this function is:

```
einteg out in laplace cpos cneg k 0.0 / 0.0 1.0
```

Differentiator element

You can also use the e element with the Laplace transformation function to model a voltage differentiator.

Consider the behavior of a differentiator. In the frequency domain, the differentiator is defined by:

$$V_{out} = ksV_{in} \tag{0.48}$$

And, in the time domain the differentiator is defined by:

$$V_{out} = k\frac{dV_{in}}{dt} \tag{0.49}$$

The transfer function for the voltage gain is:

$$H(s) = \frac{V_{out}}{V_{in}} = ks \tag{0.50}$$

This is equivalent to the Laplace transfer function with the coefficient assignments:
$a_0 = 0, a_1=k, b_0=1$

The voltage-controlled voltage source element to implement this function is:

```
ediff out in laplace cpos cneg 0.0 k / 1.0
```

Single Pole / Residue

A single pole/residue element is modeled using a transfer function which is:

$$H(s) = \frac{a_0}{b_0 + b_1 s} \tag{0.51}$$

A representative single pole element is:

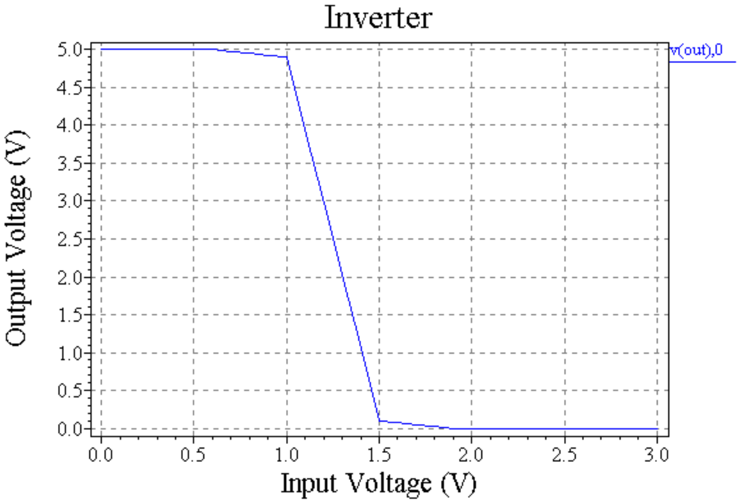
```
epole out in laplace cpos cneg 1.0 / 0.5 3.0
```

Zero-Delay Inverter Gate

The same technique used to produce the ideal amplifier, above, could be used to create a simple model of a zero-delay inverter:

```
einvert out 0 vol='table(v(in), 0.6, 5, 1, 4.9, 1.5, 0.1, 1.9, 0)'
```

This example would produce the following waveform:

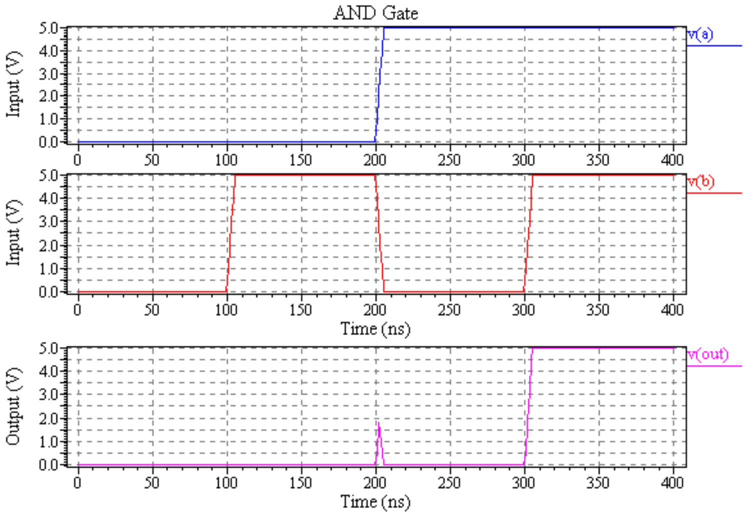


Zero-Delay AND Gate

The following example implements a simple model for a zero-delay digital AND gate:

```
eand out 0 vol='table( min(v(a),v(b)), 0, 0, 1, 0.5, 4, 4.5, 5, 5)'
```

This example would produce the following waveform:



Other logic gates can also be modeled using the same technique:

- NAND gate:

```
enand nand 0 vol='table( min(v(a),v(b)), 0, 5, 1, 4.5, 4, 0.5, 5, 0)'
```

- OR gate:

```
eor or 0 vol='table( max(v(a),v(b)), 0, 0, 1, 0.5, 4, 4.5, 5, 5)'
```

- NOR gate:

```
enor nor 0 vol='table( max(v(a),v(b)), 0, 5, 1, 4.5, 4, 0.5, 5, 0)'
```

Voltage-Controlled Oscillator (VCO)

A voltage-controlled oscillator model can be built using a combination of expression-controlled sources. The following example implements a VCO whose output voltage is a sine wave of frequency ***f0+k0*(vcontrol - vc)***, where ***vcontrol*** is the control voltage, and ***f0***, ***k0***, and ***vc*** are fixed parameters. The amplitude and offset of the VCO's output sine wave are controlled by the parameters **amp** and **offset**, respectively.

```
.param offset=2.5 amp=2.5 pi=3.141592654
.param f0=10k k0=3k vc=2.5
evco out 0 vol='offset+amp*sin(2*pi*(f0*time()+k0*v(theta)))'
gtheta theta 0 cur='vc-v(control)' chg='v(theta)'
.ic v(theta)=0
```

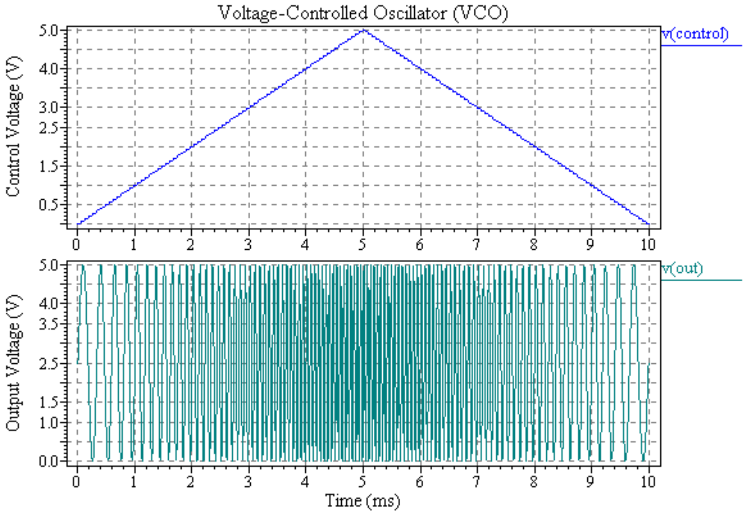
Note that a state variable called **theta** has been introduced to track the phase of the VCO. Without this additional state variable, the VCO might have been modeled using:

```
evco2 out 0 vol='offset+amp*sin(2*pi*(k0*(v(control)-vc)+f0)*time())'
```

but this would result in phase discontinuities of the VCO's output sine wave. The initial condition for **theta** is necessary to define a DC value for the phase, which would otherwise be arbitrary. The **gtheta** element implements the equation

$$\frac{d\theta}{dt} = v(control) - vc \tag{0.52}$$

The chart below shows the behavior of a VCO modeled as in the five SPICE lines above. The VCO's frequency ranges from 2.5kHz when the control voltage is zero to 17.5kHZ when the control voltage is 5V.



7 Simulation Options

This section provides a reference to the T-Spice simulation options that can be set with the **.options** (page 95) command.

Options are grouped in the following categories:

- **Accuracy and Convergence Options** (page 195)
- **Timestep and Integration Options** (page 227)
- **Model Evaluation Options** (page 246)
- **Linear Solver Options** (page 267)
- **General Options** (page 271)
- **Output Options** (page 281)
- **Probing Options** (page 306)

Accuracy and Convergence Options

absi | abstol

.options absi = *absi* where ***absi*** > 0

Default Value

1×10^{-10} A

Description

Specifies a convergence criterion that limits the total RMS (root mean square) of residual branch currents at all nodes in the circuit. The current test for convergence is applied when **kcltest** = **true**.

When the current tolerance test is applied, the system of equations is determined to be converged if:

$$\textbf{\textit{reli}} > \sqrt{\sum_{i=0}^{n} (\Delta I')^2} \qquad and \qquad \textbf{\textit{absi}} > \sqrt{\sum_{i=0}^{n} I^2}$$

where I is the residual branch current at each node. The quantity $\Delta I'$ represents the change in relative residual branch current between two consecutive iterations:

$$\Delta I' = \left(\frac{I_j}{imax_j}\right) - \left(\frac{I_{j-1}}{imax_{j-1}}\right)$$

where $imax_j$ is the largest branch current (in absolute value) flowing into the node in question in the j^{th} Newton iteration.

See Also

reli | reltol (page 224), **kcltest** (page 212)

absv | vntol

.options absv = ***absv*** where ***absv*** > 0

Default Value

1×10^{-6} V

Description

Specifies a convergence criterion limiting the absolute change in node voltage between two Newton iterations. The voltage test for convergence is applied when **kvltest** = **true**.

The voltage tolerance at each node is calculated as follows:

voltage tolerance = max(**absv**, **relv** × *V*),

where *V* is the node voltage value. If the voltage variation at each node is less than the calculated voltage tolerance for that node, then the iteration is considered to be converged.

See Also

relv (page 225), **kvltest** (page 213)

accurate

.options accurate = { true | false }

Default Value

false

Description

Triggers changes to other option settings to maximize simulation accuracy:

Option	*Default*	*Accurate*
lvltim (page 231)	1	3
numnd | itl1 (page 217)	250	500
numndset (page 218)	***numnd*** / 10 (25)	***numnd*** / 5 (100)
numns | itl6 (page 219)	50	100
numnx | itl2 (page 220)	100	200
numnxramp (page 221)	50	100
reldv | relvar (page 241)	0.35	0.3
reli | reltol (page 224)	5×10^{-4}	1×10^{-4}
rmax (page 243)	2	1

Specifying any of the above fields individually will override the value set by **accurate**.

See Also

fast (page 207), **precise** (page 222)

bypass

.options bypass = { true | false }

Default Value

true

Description

Enables or disables the diode and transistor evaluation bypass algorithm. If the terminal voltage values for a device have a relative change that is less than or equal to **bytol** since the previous evaluation, then this evaluation will be skipped, and the previous solutions used.

See Also

bytol (page 200)

bytol

.options bytol = ***bytol*** where ***bytol*** ≥ 0

Default Value

0.0

Description

Sets the relative tolerance for the bypass algorithm terminal voltage values.

See Also

bypass (page 199)

cshunt

.options cshunt = ***cshunt*** where ***cshunt*** ≥ 0.0 F

Default Value

0.0 F

Description

Adds a capacitor with the specified capacitance from each node to ground. A small ***cshunt*** value will sometimes resolve transient analysis “timestep too small” values that are caused by small, high-frequency oscillations within the circuit.

See Also

gmin (page 208), **gshunt** (page 211)

dchomotopy

.options dchomotopy = { none | source | gmin | pseudo | all }

Default Value

all

Description

Specifies the algorithm used to correct DC operating point non-convergences. When a non-convergence occurs during DC operating point analysis, the selected form of homotpy will be used to try to obtain a valid, converged solution.

Options include:

none

Do not attempt any homotopy methods.

source

Source stepping. All voltage and current sources are ramped up from zero to their final values. The smallest source step that T-Spice will take is controlled by **minsrcstep** (page 216).

gmin

Gmin stepping. T-Spice uses the g_{min} stepping algorithm to find the minimum conductance value that yields a convergent solution. The options **gmindc** and **gramp** specify a search range for the minimum required conductance, g_{min}:

$$\boldsymbol{gmindc} \le g_{min} \le \boldsymbol{gmindc} \times 10^{\boldsymbol{gramp}}$$

pseudo

Pseudotransient solution. In the pseudotransient solution method, T-Spice uses homotopy methods to approximate a solution, then removes the homotopies for the final solution. T-Spice obtains a pseudotransient solution as follows:

- T-Spice first enables pseudotransient simulation values for ***gmindc*** and ***cshunt***, which are determined internally. The source values are then ramped up to their final values.
- T-Spice then performs a time-stepping simulation. When this simulation converges, T-Spice removes the homotopy devices (***gmindc*** and ***cshunt***), one at a time, until the final solution is reached.

all

If a non-convergence occurs, T-Spice attempts homotopy methods in the following order:

- source stepping
- gmin stepping
- pseudotransient solution

As soon as a converged solution is achieved, the simulation completes without attempting the next homotopy algorithm.

Note: If you know that a specific homotopy is required to reach a convergent solution, you can reduce simulation time by setting this as the default method with **dcmethod**. Using **dcmethod** automatically skips the standard solution (no homotopy) and only attempts to calculate the solution using the method specified.

See Also

dcmethod (page 204), **minsrcstep** (page 216), **gmindc** (page 209), **gramp** (page 210)

dcmethod

.options dcmethod = {standard | source | gmin | pseudo }

Default Value

standard

Description

Specifies the default method for solving a DC operating point problem. This option is useful when you have prior knowledge that the circuit can only reach a convergent solution when a particular homotopy is required. In this case, you can reduce simulation time by setting the **dcmethod** option to the appropriate method, thus skipping attempts to solve the problem using either the standard method or the other homotopy methods. The settings for **dcmethod** are:

standard

Default setting. If a non-convergence is reached using the standard method, T-Spice will then apply the homotopy methods specified by **dchomotopy** to try to reach a convergent solution.

source

Source stepping. All voltage and current sources are ramped up from zero to their final values. The smallest source step that T-Spice will take is controlled by the **minsrcstep** option.

gmin

Gmin stepping. T-Spice finds the minimum conductance value that yields a convergent solution. . The options **gmindc** and **gramp** specify a search range for the minimum required conductance, g_{min}:

$$\textbf{\textit{gmindc}} \le g_{min} \le (\textbf{\textit{gmindc}} \times 10^{\textbf{\textit{gramp}}})$$

pseudo

Pseudotransient solution. In the pseudotransient solution method, T-Spice uses homotopy methods to approximate a solution, then removes the homotopies for the final solution. T-Spice obtains a pseudotransient solution as follows:

- T-Spice first enables pseudotransient simulation values for ***gmindc*** and ***cshunt***, which are determined internally. The source values are then ramped up to their final values.
- T-Spice then performs a time-stepping simulation. When this simulation converges, T-Spice removes the homotopy devices (***gmindc*** and ***cshunt***), one at a time, until the final solution is reached.

Note: If you do not know the best solution method for your circuit, then do not set dcmethod. Instead, set **dhomotopy** to the default value of **all**. In this case, T-Spice will automatically cycle through the homotopies as necessary to achieve convergence.

See Also

dcstep (page 205), **minsrcstep** (page 216), **gmindc** (page 209), **gramp** (page 210)

dcstep

.options dcstep = ***dcstep*** where ***dcstep*** ≥ 0

Default Value

0.0

Description

Adds a conductance across the terminals of each capacitor during DC operating point computation. If a non-zero value is specified, T-Spice computes the additional conductance for each capacitor as:

$$g = \frac{c}{\boldsymbol{dcstep}}$$

where *g* is the applied conductance and *c* is the device capacitance.

See Also

cshunt (page 201), **gmindc** (page 209)

extraiter[ations] | newtol

.options extraiter = *extraiter* where ***extraiter*** is a non-negative integer

Default Value

0

Description

Instructs T-Spice to compute the specified number of Newton solver iterative steps after convergence criteria have been met. This option is used to improve the accuracy of the solution, and is applicable to DC operating point, DC sweep, and AC simulations. For transient analysis, use the **trnewtol** option.

When **precise** = **true**, the default value of ***extraiter*** is 10.

See Also

precise (page 222), **trextraiter[ations] | trnewtol** (page 244)

fast

.options fast = { true | false }

Default Value

false

Description

Triggers changes to other options settings to maximum simulation speed:

Option	*Default*	*Fast*
absi | abstol (page 196)	1 × 10^{-10} A	5 × 10^{-10} A
bytol (page 200)	0	1 × 10^{-13}
modelmode (page 260)	**direct**	**cachetable**
reldv | relvar (page 241)	0.35	0.4
reli | reltol (page 224)	5 × 10^{-4}	1 × 10^{-3}
relq | relchgtol (page 242)	***reli*** (5 × $10^{-4)}$	***reli*** (1 × $10^{-3)}$

Specifying any of the above fields individually will override the value set by **fast**.

See Also

accurate (page 198), **precise** (page 222)

gmin

.options gmin = ***gmin*** where ***gmin*** ≥ 0

Default Value

$1 \times 10^{-12}\ \Omega^{-1}$

Description

Specifies a conductance added in parallel with all *pn* junctions during transient analaysis. When ***gmin*** > 0, T-Spice generates a ***gmin*** conductance across two terminals by adding a resistor with resistance *R*

$$R = \frac{1}{\boldsymbol{gmin}}$$

T-Spice applies the **gmin** conductance to various elements as follows:

- diode—conductance is added across the positive/negative terminals.
- —conductance is added across the base/emitter and the base/collector terminals.
- MOSFET—conductance is added across the source/bulk, drain/bulk, and the source/drain terminals.
- MESFET—conductance is added across the source/gate, drain/gate, and source/drain terminals.
- JFET—conductance is added across the source/gate, drain/gate, and source/drain terminals.

See Also

gmindc (page 209), **gshunt** (page 211)

gmindc

.options gmindc = ***gmindc*** where ***gmindc*** ≥ 0

Default Value

$1 \times 10^{-12} \Omega^{-1}$

Description

Specifies a conductance that is added in parallel with all *pn* junctions during DC operating point analysis. When ***gmindc*** > 0, T-Spice generates a ***gmindc*** conductance across two terminals by adding a resistor with resistance *R*

$$R = \frac{1}{\boldsymbol{gmindc}}$$

T-Spice applies the **gmindc** conductance to various elements as follows:

- diode—conductance is added across the positive/negative terminals.
- —conductance is added across the base/emitter and the base/collector terminals.
- MOSFET—conductance is added across the source/bulk, drain/bulk, and the source/drain terminals.
- MESFET—conductance is added across the source/gate, drain/gate, and source/drain terminals.
- JFET—conductance is added across the source/gate, drain/gate, and source/drain terminals.

Note: When a DC operating point non-convergence occurs, T-Spice can begin a g_{min} stepping algorithm to find the minimum conductance that yields a convergent solution. The g_{min} stepping algorithm is triggered when a non-convergence occurs and the **dchomotopy** option is set to **all** or **gmin**.

See Also

gramp (page 210), **dchomotopy** (page 202)

gramp

.options gramp = ***gramp*** where $0 <$ ***gramp*** $<$ (-log10 (100 × ***gmindc***))

Default Value

4

Description

Specifies the range over which the ***gmindc*** value will be swept in g_{min} stepping for DC analysis. The g_{min} stepping algorithm is triggered when a non-convergence occurs and **dchomotopy** is set to **all** or **gmin**. Together, the options **gmindc** and **gramp** specify a search range for the minimum required conductance, g_{min}:

gmindc $\leq g_{min} \leq$ ***gmindc***$\cdot 10^{\textbf{gramp}}$

T-Spice's g_{min} stepping algorithm searches the specified conductance range in two steps. First, T-Spice performs a binary search between ***gmindc*** and ***gmindc***$\cdot 10^{\textbf{\textit{gramp}}}$. T-Spice searches for the smallest value of g_{min} that results in a converged solution. T-Spice automatically ends the binary search when it reaches a Δg_{min} that is less than or equal to a factor of 10.

Starting with binary search results, T-Spice then begins reducing the value of g_{min} by a factor of 10 in each iteration. Once a non-convergence occurs, the previous convergent iteration provides the final solution.

See Also

gmindc (page 209)

gshunt

.options gshunt = *gshunt* where ***gshunt*** ≥ 0

Default Value

0.0 Ω^{-1}

Description

Specifies a conductance to be added between every node and ground. When ***gshunt*** > 0, T-Spice generates a ***gshunt*** conductance from each node to ground by adding a resistor with resistance R

$$R = \frac{1}{\boldsymbol{gshunt}}$$

See Also

cshunt (page 201), **gmin** (page 208)

kcltest

.options kcltest = {true | false }

Default Value

true

Description

Enables the current tolerance test for convergence. When the current tolerance test is applied, the system of equations is determined to be converged if:

$$\textbf{\textit{reli}} > \sqrt{\sum_{i=0}^{n} (\Delta I')^2} \qquad and \qquad \textbf{\textit{absi}} > \sqrt{\sum_{i=0}^{n} I^2}$$

where I is the residual branch current at each node. The quantity $\Delta I'$ represents the change in relative residual branch current between two consecutive iterations:

$$\Delta I' = \left(\frac{I_j}{imax_j}\right) - \left(\frac{I_{j-1}}{imax_{j-1}}\right)$$

where $imax_j$ is the largest branch current (in absolute value) flowing into the node in question in the j^{th} Newton iteration.

See Also

absi | abstol (page 196), **reli | reltol** (page 224), **kvltest** (page 213)

kvltest

.options kvltest = { true | false }

Default Value

false

Description

Enables the voltage tolerance test for convergence during transient analysis. The voltage tolerance test is always performed during DC, DC sweep, Transfer, and AC analysis. The **kvltest** option is used for enabling this test during transient analysis also.

The voltage tolerance is calculated as follows:

$\textit{voltage tolerance} = \max(\textbf{\textit{absv}}, \textbf{\textit{relv}} \times V)$,

where V is the node voltage value. If the voltage variation at each node is less than the calculated voltage tolerance for that node, then the iteration is considered to be converged.

See Also

absv | vntol (page 197), **relv** (page 225), **kcltest** (page 212)

maxdcfailures

.options maxdcfailures = n where ***n*** is a non-negative integer

Default Value

4

Description

Maximum number of non-convergence failures allowed in a DC sweep simulation before T-Spice ends processing with a “too many nonconvergences” error.

See Also

mindcratio (page 215), **numnx | itl2** (page 220)

mindcratio

.options mindcratio = *mindcratio* where $0 <$ ***mindcratio*** < 1

Default Value

1×10^{-4}

Description

Minimum fractional step size allowed in source ramping for DC sweep analysis:

$$\Delta dc_{min} = \textbf{\textit{mindcratio}} \times \Delta dc$$

where Δdc is the step size specified in the netlist **sweep** statement. If the step size falls below Δdc_{min}, T-Spice will declare a non-convergence error.

T-Spice applies source ramping when a fixed source step fails to converge. In source ramping, the source variable is gradually ramped up from the previous sweep value to the next sweep value.

See Also

maxdcfailures (page 214), **numnx | itl2** (page 220)

minsrcstep

.options minsrcstep = ***minsrcstep*** where ***minsrcstep*** > 0

Default Value

1×10^{-8}

Description

Minimum fractional step size for source stepping:

min step size = ***minsrcstep*** × (*Source value*)

In source stepping, all voltage and current sources are ramped up from zero to their final values. This allows T-Spice to find the DC operating points of difficult-to-converge circuits. Source stepping is used only in non-converging cases of initial DC operating point computations when **dchomotopy** is set to **source** or **all**.

See Also

numns | itl6 (page 219), **dchomotopy** (page 202)

numnd | itl1

.options numnd = ***numnd*** where ***numnd*** is a positive integer

Default Value

250

Description

Newton iteration limit for DC operating point computation. If a solution does not converge within ***numnd*** iterations, T-Spice applies the homotopy method specified by **dchomotopy** to attempt to reach a convergent solution. If **dchomotopy** = **none**, T-Spice declares a non-convergence error.

See Also

dchomotopy (page 202)

numndset

.options numndset = ***numndset*** where ***numndset*** is a positive integer

Default Value

numnd / 10

Description

This option is used during DC operating point computations when the user has specified node voltage guesses using the **.nodeset** command. **Numndset** is the number of Newton iterations during which the **.nodeset** nodes will be held at their user-specified voltage values. After **numndset** iterations, or when the circuit convergence criteria have been met, these node voltages are allowed to vary for the remainder of the computation

See Also

.nodeset (page 89)

numns | itl6

.options numns = ***numns*** where ***numns*** is a positive integer

Default Value

50

Description

Newton iteration limit for each source stepping attempt in DC operating point analysis. In source stepping, all voltage and current sources are ramped up from zero to their final values. This allows T-Spice to find the DC operating points of difficult-to-converge circuits. Source stepping is used only in non-converging cases of initial DC operating point computations when **dchomotopy** is set to **source** or **all**.

See Also

numnd | itl1 (page 217), **dchomotopy** (page 202)

numnx | itl2

.options numnx = *numnx* where ***numnx*** is a positive integer

Default Value

100

Description

Newton iteration limit for DC sweep computation. If a convergent solution is not reached within ***numnx*** iterations, T-Spice begins source ramping. In source ramping, the source variable is gradually ramped up from the previous sweep value to the next sweep value. The Newton iteration limit for source ramping is specified by ***numnxramp***.

See Also

numnxramp (page 221)

numnxramp

.options numnxramp = ***numnxramp*** where ***numnxramp*** is a positive integer

Default Value

50

Description

Newton iteration limit for DC sweep computation during source ramping. T-Spice applies source ramping when a fixed source step fails to converge within ***numnx*** iterations. In source ramping, the source variable is gradually ramped up from the previous sweep value to the next sweep value.

See Also

numnx | itl2 (page 220), **minsrcstep** (page 216)

precise

precise = {true | false}

Default Value

false

Description

Triggers changes to other options for extreme simulation precision. This option should only be used for single transistor characterizations, or for very simple circuits.

The following option settings will be used:

Option	*Default*	*Precise*
absdv | absvar (page 228)	0.5 V	0.3 V
absi | abstol (page 196)	1 × 10^{-10} A	5 × 10^{-11} A
absq | chgtol (page 229)	1 × 10^{-14} C	1 × 10^{-14} C
absv | vntol (page 197)	1 × 10^{-6} V	1 × 10^{-7} V
extraiter[ations] | newtol (page 206)	0	10
gmin (page 208)	1 × 10^{-12} Ω^{-1}	1 × 10^{-14} Ω^{-1}
gmindc (page 209)	1 × 10^{-12} Ω^{-1}	1 × 10^{-14} Ω^{-1}
kvltest (page 213)	**false**	**true**
lvltim (page 231)	1	3
extraiter[ations] | newtol (page 206)	0	10
numnd | itl1 (page 217)	250	500
numndset (page 218)	***numnd*** / 10 (25)	***numnd*** / 5 (100)
numns | itl6 (page 219)	50	100
numnx | itl2 (page 220)	100	200
numnxramp (page 221)	50	100
pivtol (page 269)	1 × 10^{-14}	1 × 10^{-16}
reldv | relvar (page 241)	0.35	0.25
reli | reltol (page 224)	5 × 10^{-4}	1 × 10^{-4}
relq | relchgtol (page 242)	***reli (***5 × $10^{-4)}$	***reli (***1 × $10^{-4)}$
relv (page 225)	1 × 10^{-3}	1 × 10^{-4}
rmax (page 243)	2	1
trextraiter[ations] | trnewtol (page 244)	0	1

Specifying any of the above fields individually will override the value set by **precise**.

See Also

accurate (page 198), **fast** (page 207)

reli | reltol

.options reli = ***reli*** where ***reli*** > 0

Default Value

5 × 10-4

Description

Specifies a convergence criterion limiting the relative change in total RMS branch current for all nodes in the circuit between consecutive iterations. The current test for convergence is applied when **kcltest** = **true**.

When the current tolerance test is applied, the system of equations is determined to be converged if:

$$\textbf{\textit{reli}} > \sqrt{\sum_{i=0}^{n} (\Delta I')^2} \qquad and \qquad \textbf{\textit{absi}} > \sqrt{\sum_{i=0}^{n} I^2}$$

where *I* is the residual branch current at each node. The quantity Δ*I'* represents the change in relative residual branch current between two consecutive iterations:

$$\Delta I' = \left(\frac{I_j}{imax_j}\right) - \left(\frac{I_{j-1}}{imax_{j-1}}\right)$$

where $imax_j$ is the largest branch current (in absolute value) flowing into the node in question in the j^{th} Newton iteration.

See Also

absi | abstol (page 196), **kcltest** (page 212)

relv

.options relv = ***relv*** where ***relv*** > 0

Default Value

1×10^{-3}

Description

Specifies a convergence criterion limiting the relative change in node voltage at any node in the circuit between consecutive iterations. The voltage test for convergence is applied when **kvltest** = **true**.

The voltage tolerance test uses both the **absv** and **relv** options to calculate tolerance:

voltage tolerance = max (***absv***, ***relv*** × *V*),

where *V* is node voltage. If the voltage change at each node is less than the voltage tolerance for that node, then the iteration is considered to be converged.

See Also

absv | vntol (page 197), **kvltest** (page 213)

tolmult

.options tolmult = ***tolmult*** where ***tolmult*** > 0

Default Value

1.0

Description

Multiplicative scaling factor for absolute and relative tolerance values used in the current and voltage tolerance tests for convergence. The following option values are multiplied by tolmult:

- **absi | abstol** (page 196)
- **absv | vntol** (page 197)
- **reli | reltol** (page 224)
- **relv** (page 225)

Timestep and Integration Options

absdv | absvar

.options absdv = ***absdv*** where ***absdv*** > 0

Default Value

0.5 V

Description

For transient analysis, **absdv** specifies the threshold absolute voltage change between two consecutive time steps. This quantity is used with ***reldv*** to calculate the voltage variance error measurement:

$$variance = \left|\frac{V_{n+1} - V_n}{\textbf{reldv} \cdot max(\textbf{absdv}, V_n)}\right|$$

Voltage variance is used to scale time step sizes when **lvltim** (page 231) is equal to 1, 3, or 4.

See Also

lvltim (page 231)

absq | chgtol

.options absq = ***absq*** where ***absq*** > 0

Default Value

1×10^{-14} C

Description

Minimum capacitor charge or inductor flux used to predict a timestep in the Local Truncation Error timestep algorithm (**lvltim** = 2 or 4). The **absq** option sets a floor on charge values to prevent the time step size from becoming too small.

The value of **absq** is used to calculate Local Truncation Error (LTE) as follows:

$$LTE = \frac{Q - Q_{predicted}}{\textbf{trtol} \times max(\textbf{absq}, (\textbf{relq} \times Q))}$$

When **lvltim**=2 or 4, and *LTE* > 1, T-Spice recalculates the solution at a smaller timestep. See **lvltim** (page 231) for a description of the LTE algorithm.

See Also

lvltim (page 231), **relq | relchgtol** (page 242)

ft

.options ft = *ft* where $0 < ft < 1$

Default Value

0.25

Description

Fraction by which the internal timestep is decreased if a transient analysis solution does not converge within ***numnt*** iterations. T-Spice recalculates the solution for a smaller timestep:

$$\Delta t_n = \Delta t_n' (1-\boldsymbol{ft})$$

where $\Delta t_n'$ is the size of the failed or non-converged nth timestep.

The **ft** option also determines the fraction by which the next timestep (Δt_{n+1}) is decreased if the nth timestep solution requires more than ***numntreduce*** iterations to converge:

$$\Delta t_{n+1} = \Delta t_n (1-\boldsymbol{ft}) .$$

For more information about timestep reduction algorithms, see **lvltim** (page 231).

See Also

numnt | itl4 | imax (page 238), **numntreduce | itl3** (page 239), **lvltim** (page 231)

lvltim

.options lvltim = {1 | 2 | 3 | 4}

Default Value

1

Description

Specifies the algorithm used to control timestep sizes in transient analysis simulation:

lvltim = 1	Iteration count algorithm with voltage variance test.
lvltim = 2	Local Truncation Error timestep control algorithm.
lvltim = 3	Modified iteration count with voltage variance test and timestep reversal.
lvltim = 4	Combines lvltim=1 voltage variance test, prior to the timestep, with lvltim=2 local truncation error control of timestep reversal

Iteration Count with Voltage Variance Test (lvltim = 1, 3, or 4)

All three algorithms require that the solution at each time step converge within ***numnt*** iterations. If a convergent solution is not found within ***numnt*** iterations, T-Spice recalculates the solution at a smaller timestep:

$$\Delta t_n' = \Delta t_n (1-\boldsymbol{ft})$$

where Δt_n is the original size of the nth timestep. When **lvltim** = 1, no further conditions are placed on the current timestep solution.

When **lvltim**=3, the timestep solution must also have an error value less than or equal to 1, where error is calculated as the voltage variance between time steps

$$error = \left|\frac{V_{n+1} - V_n}{max(\textbf{absdv}, (\textbf{reldv} \cdot V_n))}\right|$$

If **lvltim**=3 and the error is greater than 1, T-Spice recalculates the solution for a smaller timestep. The smaller timestep is obtained by scaling the current timestep by the error (voltage variance):

$$\Delta t_n' = \Delta t_n \cdot \left(\frac{0.9}{error}\right)$$

T-Spice continues the timestep reversal algorithm until it reaches a convergent solution within ***numnt*** iterations, such that *error* < 1.

Local Truncation Error Algorithm (lvltim = 2 or 4)

The Local Truncation Error (LTE) algorithm adjusts the timestep size according to the discretization error generated by integration. The amount of truncation error introduced by integration increases with the rate-of-change in the circuit. Local truncation error is calculated as the ratio between the error in predicted charge and the charge tolerances **relq** and **absq**. The value of **trtol** (page 245) is included as a corrective factor in the LTE calculation::

$$error = \frac{Q - Q_{predicted}}{\textbf{trtol} \cdot max((\textbf{relq} \cdot Q), \textbf{absq})}$$

If the calculated local truncation error (*LTE*) is greater than 1, then T-Spice recalculates the solution for a smaller time step:

$$\Delta t' = \Delta t \cdot \left(\frac{0.9}{error}\right)$$

Note: The LTE algorithm is error-prone in high-current devices, because the rapidly changing charge values will generate large local truncation error values. This can cause T-Spice to use extremely small timesteps, leading to slow simulations and/or “timestep too small” errors. In high-current circuits, a voltage-based timestep algorithm (**lvltim** = 1 or 3) is often the preferred choice.

Determining the Next Timestep (lvltim = 1, 2, 3, or 4)

All three algorithms use both iteration count and an error measurement to determine the size of the next timestep. After a convergent solution is found at the current timestep, T-Spice applies the following rules to determine the next timestep size:

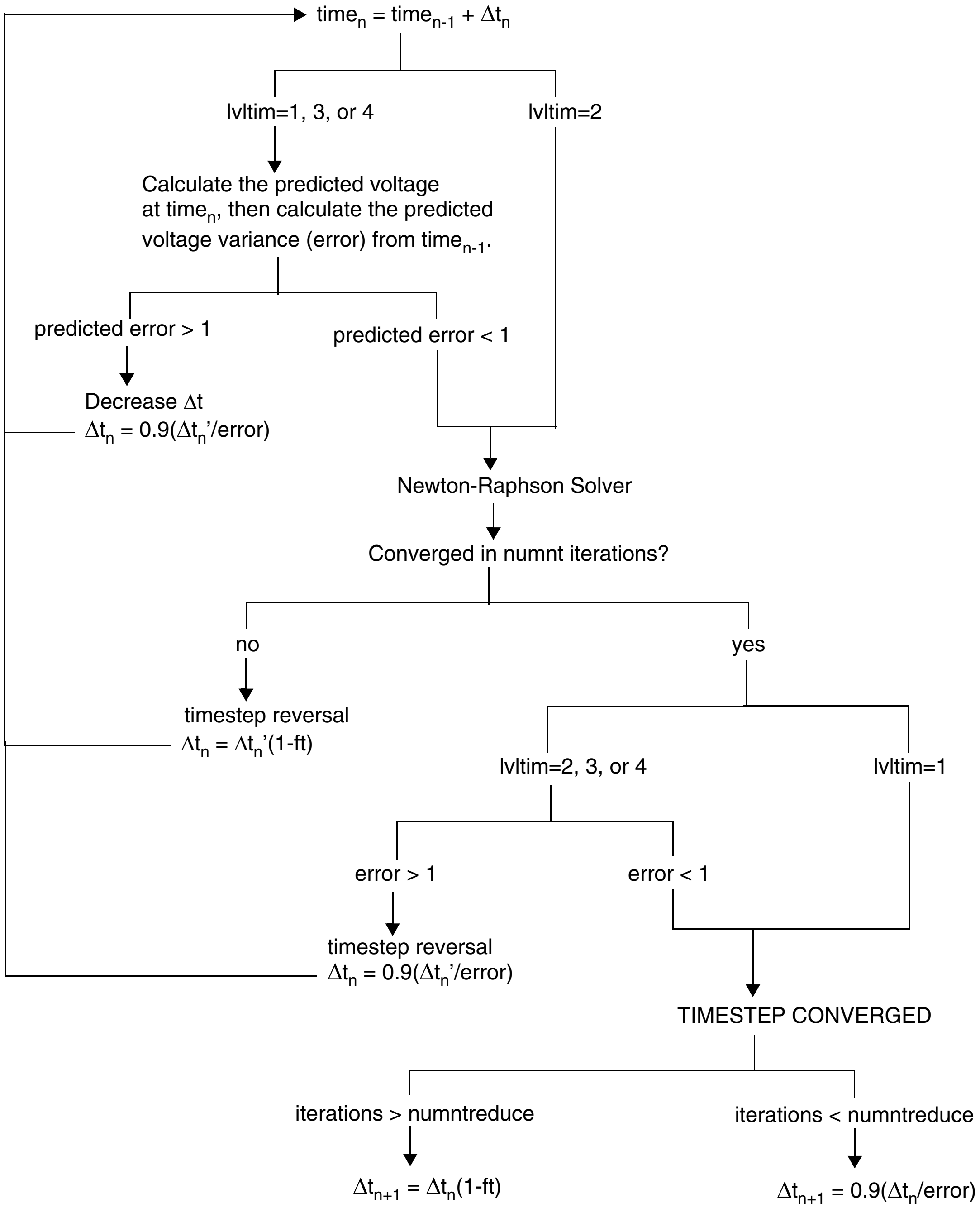
- If the solution required more than numntreduce iterations, T-Spice reduces the next timestep by the fraction ft:

$$\Delta t_{n+1} = \Delta t_n \, (1\text{-}\textbf{\textit{ft}}) \; .$$

- If the solution converged in fewer than numntreduce iterations, T-Spice uses the appropriate error measurement to scale the next timestep. (Error is equal to voltage variance when **lvltim** = 1 or 3, and LTE when **lvltim** = 2 or 4.)

$$\Delta t_{n+1} = \Delta t_n \cdot \left(\frac{0.9}{error}\right)$$

The following chart summarizes the three timestep control algorithms:


$time_n = time_{n-1} + \Delta t_n$
lvltim=1, 3, or 4
lvltim=2
Calculate the predicted voltage at $time_n$, then calculate the predicted voltage variance (error) from $time_{n-1}$.
predicted error > 1
predicted error < 1
Decrease Δt
$\Delta t_n = 0.9(\Delta t_n'/error)$
Newton-Raphson Solver
Converged in numnt iterations?
no
yes
timestep reversal
$\Delta t_n = \Delta t_n'(1-ft)$
lvltim=2, 3, or 4
lvltim=1
error > 1
error < 1
timestep reversal
$\Delta t_n = 0.9(\Delta t_n'/error)$
TIMESTEP CONVERGED
iterations > numntreduce
iterations < numntreduce
$\Delta t_{n+1} = \Delta t_n(1-ft)$
$\Delta t_{n+1} = 0.9(\Delta t_n/error)$

See Also

mintimeratio | rmin (page 236), **numnt | itl4 | imax** (page 238), **numntreduce | itl3** (page 239), **absdv | absvar** (page 228), **reldv | relvar** (page 241), **ft** (page 230)

maxord

.options maxord = {1 | 2 | 3 | 4 }

Default Value

2

Description

Maximum time integration order for variable-order Gear’s BDF calculation in transient analysis. Gear’s BDF integration is used when **method** = **gear**.

See Also

method (page 235)

method

.options method = { gear | trap }

Default Value

trap

Description

Method of numerical integration for estimating the time derivative of the system's charge components during a **.tran** simulation. Possible settings are:

gear

Variable order Gear's backward differential formula (see Gear's BDF Method on page 34).The order of integration is controlled by the option **maxord**.

trap

Trapezoidal integration. This method is faster than gear but may introduce non-physical oscillations in nodal responses. (see Trapezoidal Integration Method on page 34).

See Also

maxord (page 234)

mintimeratio | rmin

.options mintimeratio = ***mintimeratio*** where ***mintimeratio*** > 0

Default Value

1×10^{-9}

Description

Relative minimum timestep size for transient simulations. The minimum timestep for transient simulation is equal to ***mintimeratio*** × ***tstep***, where ***tstep*** is the timestep size listed on the **.tran** statement.

If the timestep size falls below ***mintimeratio*** × ***tstep***, T-Spice ends processing with a "timestep too small" error.

mu | xmu

.options mu = ***mu*** where $0 \leq$ ***mu*** ≤ 0.5

Default Value

0.5

Description

Coefficient for varying the integration between the backward Euler formula and the Trapezoidal formula. Valid values of **mu** are between 0 and 0.5, with 0 yielding a backward Euler integration, 0.5 yielding Trapezoidal integration, and intermediate values producing a hybrid integration of the specified proportional weighting. Trapezoidal integration is used when **method** = **trap.**

See Also

method (page 235)

numnt | itl4 | imax

.options numnt = ***numnt*** where ***numnt*** is a positive integer

Default Value

10

Description

Newton iteration limit for transient analysis solutions.

If a solution does not converge within ***numnt*** iterations, T-Spice recalculates the solution for a smaller time step. The fraction by which the time step is decreased after a non-convergence is specified by **ft** (page 230).

See Also

ft (page 230)

numntreduce | itl3

.options numntreduce = ***numntreduce***where ***numntreduce*** is a positive integer

Default Value

3

Description

For transient analysis, the threshold number of Newton iterations that controls the next time step size.

If more than ***numntreduce*** iterations are needed to reach a convergent solution at the nth time step, then the next time step size is reduced by the fraction ***ft***:

$$\Delta t_{n+1} = \Delta t_n (1\text{-}\boldsymbol{ft})$$

If the nth time step converges with fewer than ***numntreduce*** iterations, then T-Spice increases or decreases the next time step by scaling with a calculated error value:

$$\Delta t_{n+1} = \Delta t_n \cdot \left(\frac{0.9}{error}\right)$$

The error calculation is dependent on the algorithm specified by **lvltim** (page 231).

See Also

ft (page 230), **lvltim** (page 231)

poweruplen

.options poweruplen = ***poweruplen***

Default Value

0.1% of the total transient simulation time.

Description

Length of the powerup ramp (seconds) during powerup transient analysis.

See Also

.tran (page 137)

reldv | relvar

.options reldv = ***reldv*** where ***reldv*** > 1

Default Value

0.35

Description

For transient analysis, **reldv** specifies the maximum relative voltage change between two consecutive time steps. This quantity is used with ***absdv*** to calculate the voltage variance error measurement:

$$variance = \left|\frac{V_{n+1} - V_n}{\textbf{reldv} \cdot max(\textbf{absdv}, V_n)}\right|$$

Voltage variance is used to scale time step sizes when **lvltim** (page 231) is equal to 1, 3, or 4.

See Also

absdv | absvar (page 228), **lvltim** (page 231)

relq | relchgtol

.options relq = ***relq*** where ***relq*** > 0

Default Value

5×10^{-4}

Description

Maximum relative error in predicted capacitor charge or inductor flux. The error between predicted and actual charges is used to adjust timestep sizes in the Local Truncation Error timestep algorithm (**lvltim** = 2 or 4).

The value of **relq** is used to calculate Local Truncation Error (LTE) as follows:

$$LTE = \frac{Q - Q_{predicted}}{\mathbf{trtol} \times max(\mathbf{absq}, (\mathbf{relq} \times Q))}$$

When **lvltim**=2 or 4, and *LTE* > 1, T-Spice recalculates the solution at a smaller timestep. See **lvltim** (page 231) for a description of the LTE algorithm.

See Also

lvltim (page 231), **absq | chgtol** (page 229)

rmax

.options rmax = *rmax* where ***rmax*** ≥ 1

Default Value

2

Description

Defines the maximum allowed timestep in transient simulation, specified as a ratio. The maximum timestep is defined as:

$$\Delta t_{max} = \mathbf{rmax} \times \Delta t_{tran}$$

where Δt_{tran} is the timestep specified in the netlist **.tran** statement.

See Also

fast (page 207)**,** **accurate** (page 198)**,** **precise** (page 222)

trextraiter[ations] | trnewtol

.options trextraiter = *trextraiter* where ***trextraiter*** ≥ 0

Default Value

0

Description

Instructs T-Spice to compute the specified number of Newton solver iterative steps after convergence criteria have been met. This option is used to improve the accuracy of the solution, and is applicable to transient analysis only. For increasing the iterations of non-transient simulations use the **newtol** option.

When **precise** = **true**, the default value of ***trextraiter*** is 1.

See Also

extraiter[ations] | newtol (page 206), **precise** (page 222)

trtol

.options trtol = ***trtol***

Default Value

10

Description

Corrective factor for estimation of the local truncation error in the LTE algorithm:

$$LTE = \frac{Q - Q_{predicted}}{\textbf{trtol} \cdot max((\textbf{relq} \cdot Q), \textbf{absq})}$$

See **lvltim** (page 231) for a description of the LTE algorithm.

Note: The default value of **trtol** has been calculated to minimize error in the estimation of LTE. Changing the value of **trtol** is generally not recommended.

See Also

absq | chgtol (page 229), **relq | relchgtol** (page 242), **trtol** (page 245)

Model Evaluation Options

dcap

.options dcap = {1 | 2 }

Default Value

2

Description

Selects the model used to calculate depletion capacitance for BJTs, diodes, and MOS parasitic diodes. Possible settings are:

dcap = 1	Berkeley SPICE diode equations.
dcap = 2	Revised equations. See the device model documentation for **BJT Level 1 (Gummel-Poon)** (page 321) and **Diode** (page 351).

See Also

BJT Level 1 (Gummel-Poon) (page 321), **Diode** (page 351)

dccap

.options dccap = {true | false}

Default Value

false

Description

Controls the computation of charges and capacitances in DC simulations. Charge and capacitance values are not normally computed for DC operating point, DC sweep, and DC transfer analyses. Enabling this flag will permit the user to print out charge values for capacitors, diode, and transistors, without affecting the DC solutions.

defad

.options defad = ***defad*** where ***defad*** ≥ 0.0

Default Value

0.0 m^2

Description

Default MOSFET drain diode area.

See Also

MOSFET (m) (page 164)

defas

.options defas = *defas* where ***defas*** ≥ 0.0

Default Value

0.0 m^2

Description

Default MOSFET source diode area.

See Also

MOSFET (m) (page 164)

defl

.options defl = ***defl*** where ***defl*** ≥ 0.0

Default Value

1e-4 m

Description

Default MOSFET channel length.

See Also

MOSFET (m) (page 164)

defnrd

.options defnrd = ***defnrd*** where ***defnrd*** ≥ 0.0

Default Value

0.0

Description

Default number of diffusion squares for a MOSFET drain resistor.

See Also

MOSFET (m) (page 164)

defnrs

.options defnrs = ***defnrs*** where ***defnrs*** ≥ 0.0

Default Value

0.0

Description

Default number of diffusion squares for a MOSFET source resistor.

See Also

MOSFET (m) (page 164)

defpd

.options defpd = ***defpd*** where ***defpd*** ≥ 0.0

Default Value

0.0 m

Description

Default MOSFET drain diode perimeter.

See Also

MOSFET (m) (page 164)

defps

.options defps = *defps* where ***defps*** ≥ 0.0

Default Value

0.0 m

Description

Default MOSFET source diode perimeter.

See Also

MOSFET (m) (page 164)

deftables

.options deftables = {true | false}

Default Value

false

Description

Enables table-based model evaluation mode instead of direct model evaluation.

Note: **deftables** has been replaced with the **modelmode** option, and is documented and maintained only for purposes of backward compatibility.

See Also

modelmode (page 260)

defw

.options defw = *defw* where ***defw*** × 0.0

Default Value

1e-4 m

Description

Default MOSFET channel width.

See Also

MOSFET (m) (page 164)

deriv

.options deriv = { 0 | 1 }

Default Value

0

Description

Selects the default method for computing all device *dq/dv* and *di/dv* derivatives:

0 Computes analytical derivatives where possible. (Available for most MOSFETs, diodes, and BJTs.)

1 Only uses numerical, finite-difference derivatives.

minresistance | resmin

.options minresistance = *minresistance*

Default Value

1e-5 Ω

Description

Minimum resistance value for all resistors, including parasitics and inductor values, as well as resistors defined in the netlist. Any resistance that is specified or computed to be less than ***minresistance*** is reset to ***minresistance***.

modelmode

.options modelmode = {direct | uniform | cache}

Default Value

direct

Description

Specifies the model evaluation method used in T-Spice simulation:

direct	Direct model evaluation.
uniform | uniformtable	Table-based evaluation with uniform grid spacing.
cache | cachetable	Table-based evaluation with adaptive grid generation.

Note: Not all models support table-based evaluation.

Note: The **modelmode** option replaces the **deftables** and **adaptivegrid** options found in previous versions of T-Spice. For backward compatibility, the old options are still supported.

moscap

.options moscap = {true | false}

Default Value

false

Description

Enables automatic source/drain area/perimeter estimation for MOSFETs.

When the ACM model parameter is set to 0 (default) or 10, then:
The default AD and AS values will be **l** × **w**
The default PD and PS values will be 2 × (**l**+**w**)

See Also

MOSFET (m) (page 164)

mosparasitics

.options mosparasitics = { true | false }

Default Value

true

Description

For table-mode simulations (**modelmode=uniform** or **cache**), **mosparastics** enables explicit addition of MOSFET source/bulk drain/bulk parasitic diodes. When **mosparastics** is disabled, then the diode behavior is approximated using nonlinear capacitors, for improved speed.

See Also

modelmode (page 260), **MOSFET (m)** (page 164)

scale

.options scale = ***scale*** where ***scale*** ≥ 0.0

Default Value

1.0

Description

Scales the physical dimensions of capacitors, MESFETs, MOSFETs, and resistors. Lengths and widths are multiplied by ***scale***, and areas are multiplied by ***scale***$^{2.}$

See Also

scalm (page 264), **Capacitor (c)** (page 144), **MESFET (z)** (page 163), **MOSFET (m)** (page 164), **Resistor (r)** (page 169)

scalm

.options scalm = ***scalm*** where ***scalm*** ≥ 0.0

Default Value

1.0

Description

Default scaling factor for resistors and capacitors. The **scalm** option is overridden by setting the model parameter **scale**. The scaling factor affects lengths, widths, and drawn lengths and widths.

See Also

scale (page 263), **Capacitor** (page 348), **Resistor** (page 440)

tnom

.options tnom = *tnom*

Default Value

25 °C

Description

Nominal temperature in °C. This is the temperature at which all device and model parameters are assumed to be measured.

Note: Differences in the nominal and the simulation temperatures may account for some variations in solutions when using different SPICE simulators. Many Berkeley SPICE derivatives use 27 °C as the baseline temperature.

wl

.options wl = { true | false }

Default Value

false

Description

Reverses MOSFET length and width specifications. If **wl** = **true**, then length specifications apply to width, and width specifications apply to length.

See Also

defl (page 251), **defw** (page 257)

Linear Solver Options

linearsolver

.options linearsolver = {best | sparse | superlu }

Default Value

best

Description

Specifies how linear equations are solved. Possible settings are:

best	Allows T-Spice to select a solver based on the number of independent nodes in the system (**sparse** for less than 1000, **superlu** otherwise).
sparse	Berkeley SPICE direct solver for a system with fewer than 1000 independent nodes.
superlu	A direct solver, faster than **sparse**, for systems with more than 1000 independent nodes.

pivtol

.options pivtol = ***pivtol*** where ***pivtol*** > 0.0

Default Value

1×10^{-14}

Description

Minimum pivoting tolerance for real matrices.

zpivtol

.options zpivtol = *zpivtol* where ***zpivtol*** > 0.0

Default Value

1×10^{-6}

Description

Minimum pivoting tolerance for complex matrices.

General Options

autostop

.options autostop = { true | false }

Default Value

false

Description

Instructs T-Spice to terminate transient analysis after all **.macro /.eom** (page 77) results have been found. The autostop option does not affect transient analyses run in preview mode.

See Also

.macro /.eom (page 77), **.tran** (page 137)

casesensitive

.options casesensitive = { true | false }

Default Value

false

Description

Controls case sensitivity for names of models, subcircuits, library sections, parameters, and nodes.

Note: This option only controls case sensitivity for names that appear *after* the **.options casesensitive** command in the input file.

compatibility

.options compatibility = { spice | hspice | pspice }

Default Value

hspice

Description

Selects a compatibility mode for the input netlist syntax and for simulation settings.

T-Spice can natively read about 99% of Berkeley SPICE, HSPICE, and PSPICE input statements. There are, however, some incompatibilities and some irreconcilable syntax distinctions across the various simulators. By assigning a simulator cmpatibility mode, these input syntax ambiguities can be resolved and appropriately parsed.

In addition to providing syntax parsing support, the compatibility command sets certain simulator settings in order to mimic the behavior of the target simulator. The most notable of these settings is the default temperature, which is 25 °C in HSPICE, and 27 °C in other simulators. Other differences, such as the default MOSFET **nrd** and **nrs** values, and output format settings, may sometimes be the source of subtle or surprising differences in simulator solutions.

HSPICE compatibility mode settings:

- HSPICE compatibility mode is the default setting. Therefore, the default T-Spice settings are used.

SPICE compatibility mode settings:

- The command for assigning Berkeley SPICE compatibility, **.options compat=spice**, is equivalent to the **.options spice** command, described in **spice** (page 280).

PSPICE compatibility mode settings:

- The default temperature and **tnom** setting are 27 °C
- The **acout** (page 283) option is set to 0
- The **pwr()** function is evaluated as a signed power function in expressions

conncheck

.options conncheck = { true | false }

Default Value

true

Description

Toggles connectivity checking, which tests for common connectivity problems such as "No dc path to ground from node X" and "Node X is attached to only one device."

parhier

.options parhier={local | global}

Default value

local

Description

Establishes the scoping algorithm for selection of parameter values in a hierarchical design. Parameters can obtain their values in a number of different methods, and the varying methods must have an extablished precedence for resolving which value assignment to select when a parameter value is assigned in multiple ways or at multiple levels of hierarchy.

The ways in which parameters can be set are:

- .param statements
  example:**.param cap=600f**
- subckt parameter declaration, which establish default values
  example:**.subckt inverter in out v cap=800f**
- subckt instances (calls) with parameter assignment
  example:**xInverter n1 n2 vdd inverter cap=200f**
- .step and DC sweep simulations
  example:**.step cap 200f 1000f 200f**

The **parhier** option controls whether global **.param** statements will override the local (subcircuit) parameter assignments. With global scoping the highest level (outermost) parameter assignment will be selected. With local scoping the lowest level (innermost) parameter assignment will be selected.

The rules of scoping vary according to the parhier setting, as depicted in the following chart:

parhier=local	***parhier=global***
sweep assignments	sweep assignments
subckt call assignment	.param assignment
subckt declaration assignment	subckt call assignment
.param assignment	subckt declaration assignment

The following example demonstrates how the **parhier** setting can effect the final parameter value.

```
.param cap=600f
.subckt inverter in out v cap=800f
...
cap1 n3 gnd C=cap
.ends
xinverter1 1 5 vdd cap=400f
```

If parhier is set to **local**, then xinverter.c1 will have a capacitance of 400f. If parhier is set to **global**, then xinverter.c1 will have a capacitance of 600f.

Note: To get a listing of all parameter names, values, and the type of assignment, use the **xref** option:
.options xref=true

persist

.options persist = { 0 | 1 | 2 | 3 }

Default Value

1

Description

Instructs T-Spice to continue simulation when the specified levels of warnings or errors are generated. In the default mode, a severe warning encountered during execution causes the simulation to exit with a terminal error message.

0	Stop simulation when a warning or error occurs.
1	Ignore warning messages.
2	Ignore severe warnings.
3	Ignore error messages.

Fatal error messages cannot be ignored; these always cause the simulation to exit.

Note: The **persist** option should be used with discretion, as some warnings or errors may cause T-Spice to generate incorrect answers or to execute for a long period of time and ultimately fail.

search

.options search = *pathname*	where ***pathname*** specifies a valid directory path. If the path contains spaces, ***pathname*** must be enclosed in quotes.

Default Value

None

Description

Sets the search path for libraries and include files. T-Spice uses the search path as follows:

- If T-Spice encounters an undefined subcircuit, it automatically searches ***pathname*** for a file named ***subckt*.inc**, where ***subckt*** is the name of the subcircuit.
- If **.include** or **.lib** statements reference files that are not in the current directory, T-Spice automatically looks for these files in the ***pathname*** directory.

See Also

.if ... / .elseif ... / .else / .endif (page 71), **.lib** (page 74)

spice

.options spice = { true | false }

Default Value

false

Description

Changes other option settings to be compatible with Berkeley SPICE:

Option	*Default*	*spice*
acout (page 283)	1	0
dcap (page 247)	2	1
defnrd (page 252)	0.0	1.0
defnrs (page 253)	0.0	1.0
ingold (page 290)	0	2
tnom (page 265)	25 °C	27 °C

Specifying any of the above fields individually will override the value set by **spice.**

Output Options

acct

.options acct = { true | false }

Default Value

false

Description

Tracks and reports simulation iteration counts and other accounting statistics. This information is written to the Simulation Window, and is also recorded in the output file.

acout

.options acout = { 0 | 1 }

Default Value

1

Description

Calculation method for AC magnitude or phase differences requested in **.print** and **.probe** statements (*e.g.*, **vm(x,y)**).

- If **acout** = 0, T-Spice performs subtraction first, then calculates the magnitude or phase of the difference, *e.g.*, vm(x,y) = vm(*x*-*y*).
- If **acout** = 1, T-Spice first calculates the magnitudes or phases and then takes the difference, *e.g.*, vm(x,y) = vm(*x*) - vm(*y*).

Note: Use **acout** = 0 for compatibility with Berkeley SPICE, and **acout** = 1 for compatibility with H-Spice.

See Also

.print (page 108), **.probe** (page 122)

brief

.options brief = { true | false }

Default Value

false

Description

Turns off most of the printout which is sent to the **Simulation Status** window.

The **brief** option is equivalent to the **verbose=0** setting.

captab

.options captab = { true | false }

Default Value

false

Description

Lists the capacitances for each node in the netlist, and identifies the node with the greatest capacitance.

csdf

.options csdf = { true | false }

Default Value

false

Description

Causes T-Spice to generate output in CSDF (Common Simulation Data Format) mode for compatibility with ViewLogic Tools.

If probing has been requested in the input T-Spice netlist (see .probe on page 122), then the output probe file will be written in CSDF binary file format. If probing has not been requested, then the standard T-Spice output file will be written in CSDF text format.

dnout

.options dnout = { 0 | 1 }

Default Value

0

Description

Selects the units that T-Spice uses to measure all input and output noise spectral density magnitudes.

0	T-Spice reports noise spectral density contributions in units of Volts/sqrt(Hz).
1	T-Spice reports noise spectral density contributions in units of Volts^2/Hz.

Note: Use **dnout** = 1 for compatibility with H-Spice computations.

echo

.options echo = { true | false }

Default Value

false

Description

Causes T-Spice to print each line of input to the error log as it is read. The error log is the T-Spice GUI output window, or the specified file when the **-e *filename*** commandline option is used.

When **echo** = **true**, T-Spice lists each input next to the line number on which it occurs. For example:

```
Initializing parser with command line options
line 00001: .options verbose=2
End-of-input
line 00003: .param pres=100
line 00004: .param ptc1=0.1
line 00005: .param ptc2=0.4
line 00006:
line 00007: .options precise
line 00008:
line 00009: v1 1 0 0.01 sin(0.01 12 1e8)
...
```

expert

.options expert = { true | false }

Default Value

false

Description

Activates the printout of detailed information about non-convergent nodes and devices when a simulation, or a stage of a simulation, fails.

ingold

.options ingold = { 0 | 1 | 2 }

Default Value

0

Description

Controls the format of all real numeric data which is written to the AC Small-Signal output (**.acmodel** (page 54)) and to the device listings ()..

ingold *value*	***Format***	***Examples***
0	engineering format	-2.875u
1	g format - combined fixed and exponential format	6.234
2	e format - constant width exponential format	-1.7428e-005

For the engineering format, exponents between the value of 1e-18 and 1e15 are expressed as a single character appended to the end of the real data. Numbers which are smaller then 1e-18 or greater than 1e15 are printed in the exponential format. The characters used to express the exponential value are:

Character suffix	***Value***
T	10^{12}
G	10^{9}
X	10^{6}
K	10^{3}
m	10^{-3}
u	10^{-6}
n	10^{-9}
p	10^{-12}
f	10^{-15}
a	10^{-18}

list

.options list = { true | false }

Default Value

false

Description

Directs T-Spice to printout detailed information about every element in the netlist. The information will include nodal connectivity, the device values (resistance, capacitance, etc.), and other pertinent device parameter and device geometry settings.

maxmsg

.options maxmsg = ***maxmsg*** where ***maxmsg*** is a non-negative integer

Default Value

4

Description

Sets the maximum number of times that a duplicate warning message will be printed. A value of 0 specifies that all warning messages should be printed an unlimited number of times.

node

.options node = { true | false }

Default Value

false

Description

Prints a node cross-reference table listing each node and all the elements connected to it. Each element is listed as **element:*term***, where ***term*** identifies the element terminal as follows.

Device Type	*Terminal identifiers*
Diode	+ = anode − = diode
	B = base **C** = collector **E** = emitter **S** = substrate
MOSFET or JFET	**B** = bulk **D** = drain **S** = source **G** = gate
All other devices	**1** = terminal 1 **2** = terminal 2 etc.

nomod

.options nomod = { true | false }

Default Value

true

Description

Suppresses printout of all model parameters.

numdgt

.options numdgt = ***numdgt*** where ***numdgt*** is a non-negative integer

Default Value

4

Description

Minimum number of digits after the decimal point to be printed to the output file for each requested (**.print**) output value.

nutmeg

.options nutmeg = { true | false }

Default Value

false

Description

Generates output files in a format compatible with the *Nutmeg* graphics program.

opts

.options opts = { true | false }

Default Value

false

Description

Prints the current settings of all control options. The default behavior (**opts** = **false**) is to list only those option values that have been changed.

outputall

.options outputall = { true | false }

Default Value

false

Description

Causes all commands that list nodes, devices, or options to include internal listings. Internal nodes, devices, and options are normally hidden to the user.

The **outputall** option affects the following commands:

.options node	Includes all internal nodes.
.options list	Includes all internal nodes and devices.
.options opts	Includes all hidden and undocumented options.
.print	Tests internal devces and nodes for wildcard matching. When **.print** is used without arguments, includes all internal nodes and devices in the output.
.probe	Tests internal nodes and devices for wildcard matching. When **.probe** is used without arguments, includes all internal nodes and devices in the output.

See Also

node (page 293), **xref** (page 305), **opts** (page 297), **.print** (page 108), **.probe** (page 122)

pathnum

options pathnum = { true | false }

Default Value

false

Description

The pathnum option converts all node and element names in the output listings so that the subcircuit pathname portion of each name is converted to a number. An accompanying cross-reference will be printed to show the correspondence of these numbers to the fully qualified subcircuit pathnames.

prtdel

.options prtdel = *prtdel* where ***prtdel*** ≥ 0.0

Default Value

0.0 s

Description

Fixed time delay between output points in transient analysis. This does not affect the internal time step calculations needed to ensure solution accuracy.

prtinterp

.options prtinterp = { true | false }

Default Value

false

Description

When the **prtdel** option is set, **prtinterp** determines how solutions are calculated at the output time intervals.

- **prtinterp=0**—In addition to the internal time steps, the T-Spice simulator takes time steps at the output intervals and calculates those solutions directly.
- **prtinterp=1**—Output solutions are computed using linear interpolation of the T-Spice engine’s internal time step solutions.

statdelay

.options statdelay = ***statdelay***

Default Value

0.5 s

Description

Minimum delay in real time between updates of status display in the T-Spice user interface (GUI).

tabdelim

.options tabdelim = { true | false }

Default Value

false

Description

Toggles tab-delimited output columns.

verbose

.options verbose = { 0 | 1 | 2 | 3 | 4 }

Default Value

1

Description

Level of detail of circuit and simulation information printed to the Simulation Window.

0 Prints only the processing phase and final runtimes.

1 Prints node and device counts and major runtime statistics. Lists options whose values have been modified from the default.

2 Prints all option settings (equivalent to **opts** = **true**) and all runtime statistics (equivalent to **acct** = **true**).

3 Prints node connectivity (equivalent to **node** = **true**), lists devices (equivalent to **list** = **true**), and prints conditional statement, subcircuit, and parameter cross reference listings (equivalent to **xref** = **true**).

4 Lists hidden or internal devices, nodes, and options (equivalent to **outputall** = **true**), and lists convergence residual statistics (equivalent to **expert** = **true**).
If the input file contains **.alter** statements, then all log (listing) printouts will be given for each alter section.

See Also

opts (page 297), **acct** (page 282), **node** (page 293), **outputall** (page 298)

xref

.options xref = { true | false }

Default Value

false

Description

Generates extensive cross-referencing information listings about the input circuit and simulation commands. The information includes:

- A list of all subcirctuit instances;
- A tree outline of all conditional statements (**.if () ... .endif**);
- A listing of all parameters (**.param** definitions or subcircuit parameters) that are defined and used in the circuit.

Probing Options

binaryoutput

.options binaryoutput = { 0 | 1 | 2 | 3 }

Default Value

3

Description

Specifies the form of binary output created with **.probe** (page 122).

0	Text format.
1	Binary format with no compression.
2	Binary format using constant waveform compression.
3	Binary format using linear extrapolation compression.

probei

.options probei = { true | false }

Default Value

false

Description

Instructs T-Spice to include device terminal current values in the output data generated by **.probe** and **.print** (when used without arguments).

probeq

.options probeq = { true | false }

Default Value

false

Description

Instructs T-Spice to include device terminal charge values in the output data generated by **.probe** and **.print** (when used without arguments).

probev

.options probev = { true | false }

Default Value

false

Description

Instructs T-Spice to include node voltage values in the output data generated by **.probe** and **.print** (when used without arguments).

probefilename

```
.options probefilename = filename
```

Default Value

fname*.dat**, where ***fname is the text output filename.

Description

Specifies the filename for binary output produced by the **.probe** command.

probesdbfile

.options probesdbflle = *filename*

Default Value

None.

Description

Specifies the S-Edit design file from which the SPICE netlist was exported.

probetopmodule

.options probetopmodule = *modulename*

Default Value

None.

Description

Specifies the S-Edit design file module from which the SPICE netlist was exported.

8 Device Models

Introduction

This chapter describes the predefined analytical device models. Original documentation from the developers of the models is provided with T-Spice for further reference, in the **docs/models** folder of the installation directory, and at **Additional Model Documentation**.

BIPOLAR Level/Model Cross Reference

T-Spice supports a number of different bipolar models, which are selectable using the **level** parameter in the **.model** statement. The association between model levels and the model type is shown in the following table.

Bipolar level	*Model*	*Further Documentation*
1	Gummel-Poon	**BJT Level 1 (Gummel-Poon)** (page 321)
6	Philips Mextram	**BJT Level 6 (Mextram)** (page 341)
9	VBIC	**BJT Level 9 (VBIC)** (page 342)
10	Philips Modella Lateral PNP	**BJT Level 10 (Modella)** (page 347)

Diode Level/Model Cross Reference

The diode models which are supported by T-Spice, together with their model **level** associations, are shown in the follwowing table.

Diode level	*Model*	*Further Documentation*
1	Non-geometric Junction Diode	**Diode** (page 351)

Diode level	*Model*	*Further Documentation*
2	Fowler-Nordheim	**Diode** (page 351)
3	Geometric Junction Diode	**Diode** (page 351)
4	Philips Juncap 2	**Diode** (page 351)
200	Philips Advanced Diode	**Diode** (page 351)

JFET & MESFET Level/Model Cross Reference

The diode models which are supported by T-Spice, together with their model **level** associations, are shown in the follwowing table.

JFET/ MESFET level	*Model*	*Further Documentation*
0	Schichmann and Hodges JFET	**JFET** (page 365)
1	Curtice MESFET	**MESFET** (page 368)
2	Statz MESFET	**MESFET** (page 368)
3	Curtice GaAs MESFET	**MESFET** (page 368)

MOSFET Level/Model Cross Reference

Particular MOSFET models are selected by use of the **level** parameter in the **.model** statement. The association between model levels and the model type is shown in the following table.

Note: Some models can be referenced using more than one level number. This does not imply that the models are different. It is usually the case that the additional level is added for compatibility of input files from other simulators (Berkeley SPICE, HSPICE®, PSPICE®, etc.). For example, the EKV model can be selected as either MOSFET level 44, for SmartSpice compatibility, or as level 55, for HSPICE compatibility.

MOSFET level	*Model*	*Further documentation*
1	SPICE level 1	**MOSFET Levels 1/2/3 (Berkeley SPICE 2G6)** (page 378)
2	SPICE level 2	**MOSFET Levels 1/2/3 (Berkeley SPICE 2G6)** (page 378)
3	SPICE level 3	**MOSFET Levels 1/2/3 (Berkeley SPICE 2G6)** (page 378)
4	BSIM1	**MOSFET Levels 4 and 13 (BSIM1)** (page 395)
5	proprietary Maher-Mead model	**MOSFET Level 5 (Maher-Mead)** (page 403)
8	BSIM3 v3.3 — strict Berkeley implementation without extensions	**MOSFET Levels 8, 49 and 53 (BSIM3 Revision 3.3)** (page 410)
9	Philips MOS 9 (identical to level 50)	**MOSFET Levels 9 and 50 (Philips MOS 9)** (page 425)
11	Philips MOS 11 (identical to level 63)	**MOSFET Levels 11 and 63 (Philips MOS 11)** (page 426)
13	BSIM1	**MOSFET Levels 4 and 13 (BSIM1)** (page 395)
14	BSIM4 v4.5 (identical to level 54)	**Variables for which equations are not given here are as follows.** (page 418)
20	Philips MOS 20	**MOSFET Level 20 (Philips MOS 20)** (page 428)
28	BSIM1 with extensions	**MOSFET Level 28 (Extended BSIM1)** (page 399)
30	Philips MOS 30	**MOSFET Level 30 (Philips MOS 30)** (page 429)
31	Philips MOS 31	**MOSFET Level 31 (Philips MOS 31)** (page 430)
40	Philips MOS 40	**MOSFET Level 40 (Philips MOS 40)** (page 431)
44	EKV v2.6 (identical to level 55)	**.MOSFET Levels 44 and 55 (EKV Revision 2.6)** (page 421)
47	BSIM3 v2	**MOSFET Level 47 (BSIM3 Revision 2)** (page 405)
49	BSIM3 v3.3 with HSPICE extensions	**MOSFET Levels 8, 49 and 53 (BSIM3 Revision 3.3)** (page 410)
53	BSIM3 v3.3 with limited HSPICE extension support (ACM, tnom, etc.)	**MOSFET Levels 8, 49 and 53 (BSIM3 Revision 3.3)** (page 410)
54	BSIM4 v4.5 (identical to level 14)	**Variables for which equations are not given here are as follows.** (page 418)
55	EKV v2.6 (identical to level 44)	**MOSFET Levels 44 and 55 (EKV Revision 2.6)** (page 421)
57	BSIM3SOI v3.2	**MOSFET Level 57 (BSIM3SOI)** (page 422)
100	PSP	**MOSFET Level 100 (Penn State & Philips PSP Model)** (page 432)

Philips Model Cross Reference

T-Spice supports most of the models that are available in the Philips SiMKit compact transistor model library. The SiMKit library contains a variety of diode, bipolar, and MOSFET models, and has been made freely available to the circuit and model design community.

Note: The Philips models do not support table-based model evaluation, or HSPICE extensions (ACM, HSPICE diodes, etc.)

In order to facilitate easy selection of Philips models, an additional parameter has been added to the **.model** statement: **model=***modelname*. The *modelname* value will determine the Philips model which will be selected, and will override the **level** and **version** parameter values in the **.model** statement, if they are present. The possible values for *modelname* are listed in the following table. In this table, the **e/g** column notes whether the model is electrical or geometrical; the **thermal** column states whether the model includes self-heating equations; and the **substrate** column states whether it is a bipolar model which includes the substrate current and charge terms.

Model	*Type*	*Level*	*Modelname*	*e/g*	*Thermal*	*Substrate*
Juncap Level 1	d	1	juncap	e		
Juncap2 Level 200	d	200	juncap200	e		
Advanced Diode Level 500	d	500	dio500	e		
Modella	pnp	500	bjt500	e		yes
			bjt500t	e	yes	yes
Mextram Level 503	npn/pnp	503	bjt503	e		yes
			bjtd503	e		
Mextram Level 504	npn/pnp	504	bjt504	e		yes
			bjt504t	e	yes	yes
			bjtd504	e		
			bjtd504t	e	yes	
PSP Level 100.1	nmos/pmos	100	psp100 or psp100e	e		
		1000	psp1000 or psp100g	g		
Mos 11 Level 1100	nmos/pmos	1100	mos1100e	e		

Model	*Type*	*Level*	*Modelname*	*e/g*	*Ther mal*	*Subs trate*
			mos1100	g		
Mos 11 Level 1101	nmos/pmos	1101	mos1101e	e		
			mos1101et	e	yes	
		11010	mos11010	g		
			mos11010t	g	yes	
	(binning)	11011	mos11011	g		
			mos11011t	g	yes	
Mos 11 Level 1102	nmos/pmos	1102	mos1102e	e		
			mos1102et	e	yes	
			mos11020	g		
			mos11020t	g	yes	
	(binning)	11021	mos11021	g		
			mos11021t	g	yes	
Mos 20 Level 2001	nmos/pmos	2001	mos2001e	e		
			mos2001et	e	yes	
			mos2001	g		
			mos2001t	g	yes	
Mos 31 Level 3100	nmos/pmos	3100	mos3100	e		
			mos3100t	e	yes	
Mos 40 Level 40	nmos/pmos	40	mos40	e		
			mos40t	e	yes	
Mos 30 Level 3002	nmos/pmos	3002	mos3002	e		
Mos 9 Level 902	nmos/pmos	902	mos902e	e		
			mos902	g		
Mos 9 Level 903	nmos/pmos	903	mos903e	e		
			mos903	g		

Model Descriptions

For each model class, the following information is typically given.

- The **.model** command to be used in the input file. The **.model** command initializes the model by specifying its name, type, level, and parameter values.
- The parameters that determine the model's characteristics. These are given in a table of parameter names (as used in the code), symbols (as used in the equations), descriptions, default values, and units.
- The circuit underlying the large-signal behavior of the model.
- The analytical equations that constitute the model.

The following abbreviations and conventions are employed.

|	(Vertical bar.) In syntax paradigms, separates alternative values. In parameter tables, separates alternative names (*aliases*).
Computed.	Indicates a parameter whose value is computed, not fixed or assigned.
[n]	Indicates the NMOS value of a parameter.
[p]	Indicates the PMOS value of a parameter.
□	Symbol for *square*.
DIBL	Abbreviation for *drain-induced barrier lowering*.
LDD	Abbreviation for *lightly-doped drain*.

Constants

The following physical constants are used in parameter and equation evaluation.

Constant	*Description*	*Value*	*Units*
$\varepsilon 0$	Dielectric permittivity of vacuum	8.85421×10^{-12}	F⁄m
εox	Relative dielectric permittivity of SiO_2	3.9	—
εsi	Relative dielectric permittivity of silicon	11.7	—
ni	Intrinsic carrier concentration at 300 K	1.45×10^{16}	m^{-3}
q	Elementary electron charge	$1.6021918 \times 10^{-19}$	C
π	Pi	3.1415926	—
k	Boltzmann's constant	$1.3806226 \times 10^{-23}$	V·C⁄K
—	0 °C	273.15	K

Constant	***Description***	***Value***	***Units***
Tnom	Default nominal model temperature	Specified by **.options tnom** (*default*: 25)	°C
e	Base of natural logarithms	2.7182818	—

BJT Level 1 (Gummel-Poon)

The level 1 bipolar junction transistor model uses a modified version of the Gummel-Poon charge-control model that was implemented in the original SPICE. The model simplifies to the Ebers-Moll model when certain parameters are not specified. It also includes high-bias and temperature effects.

Note: Gummel-Poon is the default model which will be used for all bipolar models (type **npn** or **pnp**) which do not have the **level** model parameter set.

Parameters

The BJT model uses the following syntax.

```
.model name npn|pnp [level=1] [parameters]
```

The following tables describe all of the Gummel-Poon BJT parameters that T-Spice supports. Parameters that are not specified in the Ebers-Moll model are marked with an asterisk (*).

DC Current Parameters

Parameter (alias)	*Symbol*	*Description*	*Default*	*Units*
level		Model selector	1.0	—
subs		Substrate connection selector	1 (npn); -1 (pnp)	—
update		Equation selector for base charge.	0	—
dcap		Equation selector for depletion capacitance.	2	—
bf	βf	Ideal forward maximum current gain.	100.0	—
br	β_r	Ideal reverse maximum current gain.	1.0	—
ibc	I_{bc}	Reverse saturation current between base and collector.	0.0	A
ibe	I_{be}	Reverse saturation current between base and emitter.	0.0	A
iss	I_{ss}	Reverse saturation current between bulk and collector for vertical geometry , or between bulk and base for lateral geometry	0.0	A
is	I_s	Transport saturation current.	1.0×10^{-16}	A
c2 (jle)	C_2	Non-ideality factor for base-emitter leakage saturation current.*	0.0	—
c4	C_4	Non-ideality factor for base-collector leakage saturation current.*		—
ise	I_{se}	Base-emitter leakage saturation current.*	**c2**×**is**	A
isc	I_{sc}	Base-collector leakage saturation current.*	**c4**×**is**	A
nf	η_f	Forward current emission coefficient.*	1.0	—
nr	η_r	Reverse current emission coefficient.*	1.0	—
ns	η_s	Substrate current emission coefficient.	1.0	—
nc (nlc)	η_c	Base-collector leakage emission coefficient.*	2.0	—
ne (nle)	η_e	Base-emitter leakage emission coefficient.*	1.5	—
expli	EXPLI	Current explosion model parameter. The PN junction characteristics above the explosion current area are linear, with the slope at the explosion point.	1.0×10^{15}	A

Base Charge Parameters

Base charge parameters are not specified for the Ebers-Moll model.

Parameter (alias)	*Symbol*	*Description*	*Default*	*Units*
vaf (vbf)	*Vaf*	Forward early voltage.*	0.0 (indicates infinite value)	V
var (vb, vbb)	V_{ar}	Reverse early voltage.*	0.0 (indicates infinite value)	V
ikf (ik, jbf)	I_{kf}	Corner for forward Beta high current roll-off.*	0.0 (indicates infinite value)	A
ikr (jbr)	I_{kr}	Corner for reverse Beta high current roll-off.*	0.0 (indicates infinite value)	A
nkf	η_{kf}	Exponent for high current Beta roll-off.*	0.5	

Parasitic Resistor Parameters

Parameter (alias)	*Symbol*	*Description*	*Default*	*Units*
irb (jrb,iob)	*Irb*	Base current, where base resistance falls halfway between r_b and *rbm*.*	0.0 (indicates infinite value)	A
rb	r_b	Base resistance.*	0.0	Ω
rbm	r_{bm}	Minimum high current base resistance.*	**rb**	Ω
re	r_e	Emitter resistance.	0.0	Ω
rc	r_c	Collector resistance.	0.0	Ω

Parasitic Capacitance Parameters

Parasitic capacitances are only used in the direct evaluation method of circuit analysis. They do not contribute to analysis in table mode.

Parameter	*Symbol*	*Description*	*Default*	*Units*
cbcp	*CBCP*	External base-collector constant capacitance.	0.0	F
cbep	CBEP	External base-emitter constant capacitance.	0.0	Φ
ccsp	CCSP	External collector-substrate (vertical) or base-substrate (lateral) constant capacitance.	0.0	Φ

Junction Capacitance Parameters

Parameter (alias)	***Symbol***	***Description***	***Default***	***Units***
cjc	*Cjc*	Base-collector zero-bias depletion capacitance.	0.0	F
cje	C_{je}	Base-emitter zero-bias depletion capacitance.	0.0	Φ
cjs (ccs, csub)	C_{js}	Zero-bias collector-substrate capacitance.	0.0	Φ
fc	FC	Coefficient for forward bias depletion capacitance.	0.5	
mjc (mc)	m_{jc}	Base-collector junction exponent (grading factor).	0.33	
mje (me)	m_{je}	Base-emitter junction exponent (grading factor).	0.33	
mjs (esub)	m_{js}	Substrate junction exponent (grading factor).	0.5	
vjc (pc)	V_{jc}	Base-collector built-in potential.	0.75	V
vje (pe)	V_{je}	Base-emitter built-in potential.	0.75	V
vjs (psub)	V_{js}	Substrate junction built-in potential.	0.75	V
xcjc (cdis)	X_{cjc}	Internal base fraction of base-collector depletion capacitance.	1.0	

Transit time parameters

Parameter (alias)	***Symbol***	***Description***	***Default***	***Units***
ptf	P_{tf}	Frequency multiplier to determine excess phase.	0.0	deg.
tf	τ_f	Base forward transit time.	0.0	s
tr	τ_r	Base reverse transit time.	0.0	s
vtf	V_{tf}	Voltage for V_{bc} dependence of τ_f.	0.0 (indicates infinite value)	ς
xtf	X_{tf}	Coefficient for bias dependence of τf.	0.0	
itf (jtf)	*Itf*	Parameter for high-current effect on τ_f.	0.0	A

Noise Parameters

Parameter	***Symbol***	***Description***	***Default***
af	*AF*	Flick noise exponent.	1.0
kf	KF	Flick noise exponent.	0.0

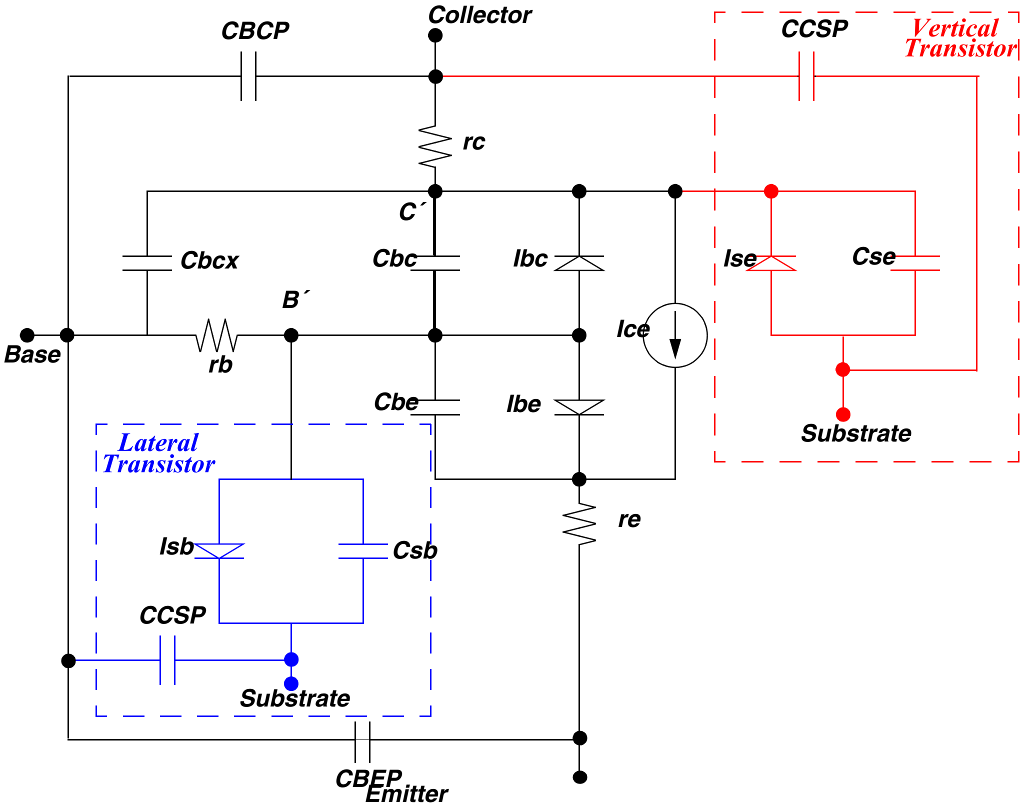
Temperature Effect Parameters

Parameter ***(alias)***	***Symbol***	***Description***	***Default***	***Units***
tnom (tref)	T_{nom}	Nominal temperature	global **tnom** (25.0)	deg
tlev	tlev	Temperature equation selector	0	
tlevc	tlevc	Temperature equation selector for junction capacitances and potentials	0	
bex	*Bex*	*V0* temperature exponent. (Level 2 only.)	2.42	
bexv	B_{exv}	*Rc* temperature exponent. (**tlev=2** only.)	1.90	
ctc	C_{tc}	Temperature coefficient for zero-bias base-collector capacitance.	0.0	1/deg
cte	C_{te}	Temperature coefficient for zero-bias base-emitter capacitance.	0.0	1/deg
cts	C_{ts}	Temperature coefficient for zero-bias substrate capacitance.	0.0	1/δεγ
eg	$E_g(0)$	Energy gap at 0°K		eV
		for **tlev** = **2** (default):	1.16	
		for **tlev** = **0**, **1**, or 3:	1.11	
gap1	GAP1	Coefficient in energy gap temperature equation	7.02×10^{-4}	eV/deg
gap2	GAP2	Coefficient in energy gap temperature equation	1108.0	deg
tbf1	TBF_1	First order and	0.0	1/deg
tbf2	TBF_2	second order temperature coefficients for β_f.	0.0	$1/deg^2$
tbr1	TBR_1	First order and	0.0	1/deg
tbr2	TBR_2	second order temperature coefficients for βr.	0.0	$1/deg^2$
tikf1	$TIKF_1$	First order and	0.0	1/deg
tikf2	$TIKF_2$	second order temperature coefficients for *Ikf*.	0.0	$1/deg^2$
tikr1	$TIKR_1$	First order and	0.0	1/deg
tikr2	$TIKR_2$	second order temperature coefficients for *Ikr*.	0.0	$1/deg^2$
tirb1	$TIRB_1$	First order and	0.0	1/deg
tirb2	$TIRB_2$	second order temperature coefficients for *Irb*.	0.0	$1/deg^2$

Parameter (alias)	***Symbol***	***Description***	***Default***	***Units***
tisc1	TISC$_1$	First order and	0.0	1/deg
tisc2	TISC$_2$	second order temperature coefficients for *Isc*. (**tlev=3** only.)	0.0	1/deg^2
tis1	TIS$_1$	First order and	0.0	1/deg
tis2	TIS$_2$	second order temperature coefficients for *Is* or *Ibe* and *Ibc*. (**tlev=3** only).	0.0	1/deg^2
tise1	TISE1	First order and	0.0	1/deg
tise2	TISE2	second order temperature coefficients for *Ise*. (**tlev=3** only)	0.0	1/deg^2
tiss1	TISS$_1$	First order and	0.0	1/deg
tiss2	TISS$_2$	second order temperature coefficients for *Iss* (**tlev=3** only).	0.0	1/deg^2
titf1	TITF$_1$	First order and	0.0	1/deg
titf2	TITF$_2$	second order temperature coefficients for *Itf*.	0.0	1/deg^2
tmjc1	TMJC$_1$	First order and	0.0	1/deg
tmjc2	TMJC$_2$	second order temperature coefficients for *mjc*.	0.0	1/deg^2
tmje1	TMJE$_1$	First order and	0.0	1/deg
tmje2	TMJE$_2$	second order temperature coefficients for *mje*.	0.0	1/deg^2
tmjs1	TMJS$_1$	First order and	0.0	1/deg
tmjs2	TMJS$_2$	second order temperature coefficient for *mjs*.	0.0	1/deg^2
tnc1	TNC$_1$	First order and	0.0	1/deg
tnc2	TNC$_2$	second order temperature coefficients for ηc.	0.0	1/deg^2
tne1	TNE$_1$	First order and	0.0	1/deg
tne2	TNE$_2$	second order temperature coefficients for ηe.	0.0	1/deg^2
tnf1	TNF$_1$	First order and	0.0	1/deg
tnf2	TNF$_2$	second order temperature coefficients for η_f.	0.0	1/deg^2
tnr1	TNR$_1$	First order and	0.0	1/deg
tnr2	TNR$_2$	second order temperature coefficients for ηr.	0.0	1/deg^2
tns1	TNS$_1$	First order and	0.0	1/deg
tns2	TNS$_2$	second order temperature coefficients for ηs.	0.0	1/deg^2
trb1	TRB$_1$	First order and	0.0	1/deg
trb2	TRB$_2$	second order temperature coefficients for *rb*.	0.0	1/deg^2
trc1	TRC$_1$	First order and	0.0	1/deg
trc2	TRC$_2$	second order temperature coefficients for *rc*.	0.0	1/deg^2
tre1	TRE$_1$	First order and	0.0	1/deg
tre2	TRE$_2$	second order temperature coefficients for *re*.	0.0	1/deg^2
trm1	TRM$_1$	First order and	trb1	1/deg
trm2	TRM$_2$	second order temperature coefficients for *rbm*.	trb2	1/deg^2
ttf1	TTF$_1$	First order and	0.0	1/deg
ttf2	TTF$_2$	second order temperature coefficients for τf.	0.0	1/deg^2
ttr1	TTR$_1$	First order and	0.0	1/deg
ttr2	TTR$_2$	second order temperature coefficients for τr.	0.0	1/deg^2

Parameter (alias)	*Symbol*	*Description*	*Default*	*Units*
tvaf1 **tvaf2**	TVF_1 TVF_2	First order and second order temperature coefficients for *Vaf*.	0.0 0.0	1/deg $1/deg^2$
tvar1 **tvar2**	TVR_1 TVR_2	First order and second order temperature coefficients for *Var*.	0.0 0.0	1/deg $1/deg^2$
tvjc	T_{vjc}	Temperature coefficient for *Vjc*.	0.0	V/deg
tvje	T_{vje}	Temperature coefficient for V_{je}.	0.0	V/deg
tvjs	T_{vjs}	Temperature coefficient for *Vjs*.	0.0	V/deg
xtb **(tb, tcb)**	X_{tb}	Forward and reverse Beta temperature exponent.	0.0	
xti	X_{ti}	Saturation current temperature exponent. (Use *Xti*=3.0 for Silicon diffused junction, and *Xti*=2.0 for Schottky barrier diode.)	3.0	

Large-Signal Model



Equations

The Gummel-Poon model includes four “second-order” current effects:

- The low-current drop in current gain β is due to extra components of base current *IB*. The parameters I_{se}, η_e (the low current drop in β_f) and I_{sc}, η*c* (the low current drop in β_r) describe this effect, and two non-ideal diodes are included in the Large-Signal Model, above as shown.

- Base-width modulation, affecting the BJT output conductance, is described by V_{ar} (reverse Early voltage) and V_{af} (forward Early voltage).
- High-level injection, modifying the slope of the log(IC) vs. V_{be} characteristic, is described by two "knee" currents, I_{kf} and I_{kr}, for the forward and reverse regions of operation.
- Base resistance is current dependent and is modeled by a combination of r_b, r_{bm}, and I_{rb}.

The Gummel-Poon large-signal model for transient simulations is topologically identical to the Ebers-Moll large-signal model except for the inclusion of the distributed base-collector capacitance. In addition, the effect of τ_f modulation (transit time) is handled.

Geometry Consideration

The BJT model has two geometric configurations based on physical layout: vertical and lateral. The geometric configuration is specified by the substrate connection selector parameter, **subs**. Set **subs** equal to 1 to specify vertical geometry or to −1 to specify lateral geometry. The default value of **subs** is 1 for **npn** transistors and −1 for **pnp** transistors.

The instance parameters **area**, **areab**, **areac**, and **M** specify geometric scaling for current and charge values in the Gummel-Poon BJT. (See BJT (q) on page 142 for more information about these parameters.)

Scaling: Both Vertical and Lateral Geometry

$$\begin{aligned} I_{s_{eff}} &= area \cdot M \cdot I_s \\ I_{be_{eff}} &= area \cdot M \cdot I_{be} \\ I_{se_{eff}} &= area \cdot M \cdot I_{se} \\ I_{kf_{eff}} &= area \cdot M \cdot I_{kf} \\ I_{kr_{eff}} &= area \cdot M \cdot I_{kr} \\ I_{rb_{eff}} &= area \cdot M \cdot I_{rb} \\ I_{ss_{eff}} &= area \cdot M \cdot I_{ss} \quad \text{\textit{if both} } I_{be} \text{ \textit{and} } I_{bc} \text{ \textit{are NOT specified}} \\ I_{tf_{eff}} &= area \cdot M \cdot I_{tf} \end{aligned} \tag{0.53}$$

$$EXPLI_{eff} = area \cdot M \cdot EXPLI \tag{0.54}$$

$$\begin{aligned} C_{bcp_{eff}} &= area \cdot M \cdot C_{bcp} \\ C_{bep_{eff}} &= area \cdot M \cdot C_{bep} \\ C_{csp_{eff}} &= area \cdot M \cdot C_{csp} \\ C_{je_{eff}} &= area \cdot M \cdot C_{je} \end{aligned} \tag{0.55}$$

$$\begin{aligned} r_{b_{eff}} &= \frac{r_b}{area \cdot M} \\ r_{bm_{eff}} &= \frac{r_{bm}}{area \cdot M} \\ r_{e_{eff}} &= \frac{r_e}{area \cdot M} \\ r_{c_{eff}} &= \frac{r_c}{area \cdot M} \end{aligned} \tag{0.56}$$

Scaling: Vertical Geometry (**subs=1**)

$$\begin{aligned} I_{bc_{eff}} &= areab \cdot M \cdot I_{bc} \\ I_{sc_{eff}} &= areab \cdot M \cdot I_{sc} \\ I_{ss_{eff}} &= areab \cdot M \cdot I_{ss} \quad \text{\textit{if both} } I_{be} \text{ \textit{and} } I_{bc} \text{ \textit{are specified}} \\ C_{js_{eff}} &= areab \cdot M \cdot C_{js} \\ C_{jc_{eff}} &= areab \cdot M \cdot C_{jc} \end{aligned} \tag{0.57}$$

Scaling: Lateral Geometry (**subs=-1**)

$$\begin{aligned} I_{bc_{eff}} &= areac \cdot M \cdot I_{bc} \\ I_{sc_{eff}} &= areac \cdot M \cdot I_{sc} \\ I_{ss_{eff}} &= areac \cdot M \cdot I_{ss} \qquad \textit{if both } I_{be} \textit{ and } I_{bc} \textit{ are specified} \\ C_{js_{eff}} &= areac \cdot M \cdot C_{js} \\ C_{jc_{eff}} &= areac \cdot M \cdot C_{jc} \end{aligned} \tag{0.58}$$

Current Equations (Level 1)

The following current equations are used when **level**=1, specifying that either the Gummel-Poon or Ebers-Moll model will be used. T-Spice chooses appropriate current equations according to which parameters were specified. The reverse saturation currents between base and collector and between base and emitter (I_{bc} and I_{be}) are optional parameters. If they are not specified, T-Spice uses the transport saturation current I_s in current equations.

There are no explicit switches between the different regions of device operation. Equations (0.59) through (0.71) describe currents in the normal active, inverse, saturation, and cut-off regions of operation. This means that the β roll-off at high collector current is slightly less pronounced in T-Spice.

Base Charge Equations

In order to determine BJT currents, T-Spice must first calculate base charge. There are two sets of base charge equations; you can specify which equations T-Spice will use by changing the model selector **update**.

The base charge q_b is calculated from the following equation:

$$q_b = \frac{q_1}{2}(1 + (1 + 4q_2)^{\eta_{kf}}), \tag{0.59}$$

where q_2 and q_1 are calculated as follows.

$$q_2 = \frac{I_{se_{eff}}}{I_{kf_{eff}}}\left(e^{\frac{V_{be}}{\eta_f V_t}} - 1\right) + \frac{I_{se_{eff}}}{I_{kr_{eff}}}\left(e^{\frac{V_{bc}}{\eta_r V_t}} - 1\right) \tag{0.60}$$

The thermal voltage V_t is defined as:

$$V_t = kT/q, \tag{0.61}$$

where k is Boltzmann’s constant, T is temperature, and q is the elementary electron charge.

If **update=1** *and* $\frac{V_{bc}}{V_{af}} + \frac{V_{be}}{V_{ar}} \geq 0$, then

$$q_1 = 1 + \frac{V_{bc}}{V_{af}} + \frac{V_{be}}{V_{ar}}. \tag{0.62}$$

If **update=0** $or\ \frac{V_{bc}}{V_{af}} + \frac{V_{be}}{V_{ar}} < 0$, then

$$q_1 = \left(1 - \frac{V_{bc}}{V_{af}} - \frac{V_{be}}{V_{ar}}\right)^{-1}. \tag{0.63}$$

Collector and Base Current Equations

If only the saturation current I_s is specified, T-Spice uses the following equations to calculate collector current (I_c) and base current (I_b):

$$I_c = \frac{I_{s_{eff}}}{q_b}\left(e^{\frac{V_{BE}}{\eta_f V_t}}\right) - \frac{I_{s_{eff}}}{q_b}\left(e^{\frac{V_{BC}}{\eta_r V_t}}\right) - \frac{I_{s_{eff}}}{\beta_r}\left(e^{\frac{V_{bc}}{\eta_r V_t}} - 1\right) - I_{sc_{eff}}\left(e^{\frac{V_{bc}}{\eta_c V_t}} - 1\right) \tag{0.64}$$

$$I_b = \frac{I_{s_{eff}}}{\beta_f}\left(e^{\frac{V_{be}}{\eta_f V_t}} - 1\right) - \frac{I_{s_{eff}}}{\beta_r}\left(e^{\frac{V_{bc}}{\eta_r V_t}} - 1\right) + I_{se_{eff}}\left(e^{\frac{V_{be}}{\eta_e V_t}} - 1\right) + I_{sc_{eff}}\left(e^{\frac{V_{bc}}{\eta_c V_t}} - 1\right) \tag{0.65}$$

If I_{be} and I_{bc} are both specified, T-Spice uses the following current equations:

$$I_c = \frac{I_{be_{eff}}}{q_b}\left(e^{\frac{V_{be}}{\eta_f V_t}} - 1\right) - \frac{I_{bc_{eff}}}{q_b}\left(e^{\frac{V_{bc}}{\eta_r V_t}} - 1\right) - \frac{I_{bc_{eff}}}{\beta_r}\left(e^{\frac{V_{bc}}{\eta_r V_t}} - 1\right) - I_{sc_{eff}}\left(e^{\frac{V_{bc}}{\eta_c V_t}} - 1\right) \tag{0.66}$$

$$I_b = \frac{I_{be_{eff}}}{\beta_f}\left(e^{\frac{V_{be}}{\eta_f V_t}} - 1\right) + \frac{I_{bc_{eff}}}{\beta_r}\left(e^{\frac{V_{bc}}{\eta_r V_t}} - 1\right) + I_{se_{eff}}\left(e^{\frac{V_{be}}{\eta_e V_t}} - 1\right) + I_{sc_{eff}}\left(e^{\frac{V_{bc}}{\eta_c V_t}} - 1\right) \tag{0.67}$$

where q_b is the normalized base charge, β_r and β_f are the ideal maximum reverse and forward current gains, η_f and η_r are the forward and reverse current emission coefficients, and η_c and η_e are the base-collector and emitter-collector leakage emission coefficients.

The last two terms in the I_b expression represent recombination in the base-emitter and base-collector surfaces and depletion regions. The last term in the I_c expression represents recombination in the base-collector junction.

The emitter current, I_e, is simply the sum of base and collector currents:

$$I_e = I_b + I_c. \tag{0.68}$$

Excess Phase Equation

Excess phase represents the extra degrees of phase delay introduced by the BJT as a function of frequency, due to the distributed phenomena in the base region. T-Spice calculates excess phase in AC and transient analyses of the BJT. At any given frequency, the phase multiplier parameter **ptf** determines the relationship between excess phase and the base forward transit time, **tf**.

$$excess\ phase = \left(2\pi \cdot P_{tf} \cdot \frac{\tau_f}{360}\right) \cdot 2\pi f \tag{0.69}$$

In AC analysis, T-Spice applies excess phase as a linear phase delay in the transconductance generator, g_m. See the small-signal reference for BJT Level 1 (Gummel-Poon) on page 450 for a description of g_m.

In transient analysis, excess phase affects how T-Spice calculates collector current (I_c). When excess phase is present (*i.e.*, **ptf≠0**), T-Spice calculates the collector current as a cumulative function using time step dependent backward Euler integration. Otherwise, if **ptf** is not specified, collector current is simply a function of the current time step.

Substrate Current Equations

Substrate current is defined according to the geometric configuration of the BJT. The substrate current flows from substrate to collector for vertical BJTs (**subs=1**) and from substrate to base for lateral BJTs (**subs=–1**).

For vertical transistors (**subs=1**),

$$\begin{aligned} I_{sc} &= I_{ss_{eff}}\left(e^{\frac{V_{sc}}{\eta_s V_t}} - 1\right) && when\ \ V_{sc} > -10 \cdot \eta_s \cdot V_t\ , \\ I_{sc} &= -I_{ss_{eff}} && when\ \ V_{sc} \le -10 \cdot \eta_s \cdot V_t \end{aligned} \tag{0.70}$$

where I_{sseff} is the effective reverse saturation current between bulk and collector.

For lateral transistors (**subs=–1**),

$$\begin{aligned} I_{bs} &= I_{ss_{eff}}\left(e^{\frac{V_{bs}}{\eta_s V_t}} - 1\right) && when\ \ V_{bs} > -10 \cdot \eta_s \cdot V_t\ , \\ I_{bs} &= -I_{ss_{eff}} && when\ \ V_{bs} \le -10 \cdot \eta_s \cdot V_t \end{aligned} \tag{0.71}$$

where I_{sseff} is the effective reverse saturation current between bulk and base.

Variable Base Resistance Equations

The following equations describe the calculation of the base resistance, r_{bb}. Base resistance varies with current and depends on two parameters, the low-current maximum resistance (r_b) and the high-current minimum resistance (r_{bm}). There are two ways that T-Spice can calculate r_{bb}, either with or without the parameter I_{rb}. I_{rb} is the base current that occurs when r_{bb} is equal to 0.5×(r_b–r_{bm}), or the midpoint between minimum and maximum resistance values.

If *Irb* is not specified, T-Spice uses the following equation:

$$r_{bb} = r_{bm_{eff}} + \frac{r_{b_{eff}} - r_{bm_{eff}}}{q_b} \tag{0.72}$$

If I_{rb} is specified,

$$r_{bb} = r_{bm_{eff}} + 3(r_{b_{eff}} - r_{bm_{eff}})\frac{\tan(z) - z}{z\tan(z)\tan(z)}\ , \tag{0.73}$$

where

$$z = \frac{-1 + \sqrt{1 + \frac{144 \cdot I_b}{\pi^2 \cdot I_{rb_{eff}}}}}{\frac{24}{\pi^2} \cdot \sqrt{\frac{I_b}{I_{rb_{eff}}}}}. \tag{0.74}$$

Capacitance Equations

Base-Emitter Capacitance

The total base-emitter capacitance consists of contributions from diffusion capacitance and depletion capacitance:

$$C_{be} = C_{be_{diff}} + C_{be_{dep}} \tag{0.75}$$

The diffusion capacitance is determined as follows:

$$\begin{aligned} C_{be_{diff}} &= \frac{\partial}{\partial V_{beb}}\left(\tau_f \cdot \frac{i_{be}}{q_b}\right) && when \quad i_{be} \le 0 \\ C_{be_{diff}} &= \frac{\partial}{\partial V_{be}}\left(\tau_f \cdot (1 + \arg\tau_f) \cdot \frac{i_{be}}{q_b}\right) && when \quad i_{be} > 0 \end{aligned} \tag{0.76}$$

where

$$\arg\tau_f = X_{tf}\left(\frac{i_{be}}{i_{be} + I_{tf}}\right)^2 \cdot e^{\frac{V_{bc}}{1.44 \cdot V_{tf}}} \tag{0.77}$$

and

$$i_{be} = I_{s_{eff}}\left(e^{\frac{qV_{be}}{\eta_f kT}} - 1\right). \tag{0.78}$$

T-Spice supports two different models for depletion capacitance. You can specify the depletion capacitance equations using the **dcap** model selector.

If **dcap=1**,

$$\begin{aligned} C_{be_{dep}} &= C_{je_{eff}}\left(1 - \frac{V_{be}}{V_{je}}\right)^{-m_{je}} && when \quad V_{be} < FC \cdot V_{je} \\ C_{be_{dep}} &= C_{je_{eff}} \cdot \frac{1 - FC(1 + m_{je}) + m_{je} \cdot \frac{V_{be}}{V_{je}}}{(1 - FC)^{1 + m_{je}}} && when \quad V_{be} \ge FC \cdot V_{je} \end{aligned} \tag{0.79}$$

If **dcap=2** (default),

$$\begin{aligned} C_{be_{dep}} &= C_{je_{eff}}\left(1-\frac{V_{be}}{V_{je}}\right)^{-m_{je}} && when \quad V_{be} < 0 \\ C_{be_{dep}} &= C_{je_{eff}}\left(1+m_{je}\cdot\frac{V_{be}}{V_{je}}\right) && when \quad V_{be} \ge 0 \end{aligned} \tag{0.80}$$

Base-Collector Capacitance

The total base-collector capacitance consists of contributions from diffusion capacitance and depletion capacitance:

$$C_{bc} = C_{bc_{diff}} + C_{bc_{dep}} \tag{0.81}$$

The base-collector diffusion capacitance is determined as follows:

$$C_{bc_{diff}} = \frac{\partial}{\partial V_{bc}}(\tau_r \cdot i_{bc}) \tag{0.82}$$

where

$$i_{bc} = I_{s_{eff}}\left(e^{\frac{V_{bc}}{\eta_r V_t}} - 1\right). \tag{0.83}$$

T-Spice offers two different models for base-collector depletion capacitance. Use **dcap** to select a set of equations.

For **dcap=1**:

$$\begin{aligned} C_{bc_{dep}} &= X_{cjc}\cdot C_{jc_{eff}}\left(1-\frac{V_{bc}}{V_{jc}}\right)^{-m_{jc}} && when \quad V_{bc} < FC\cdot V_{jc} \\ C_{bc_{dep}} &= X_{cjc}\cdot C_{jc_{eff}}\cdot\frac{1-FC(1+m_{jc})+m_{jc}\cdot\frac{V_{bc}}{V_{jc}}}{(1-FC)^{1+m_{jc}}} && when \quad V_{bc} \ge FC\cdot V_{jc} \end{aligned} \tag{0.84}$$

For **dcap=2**:

$$\begin{aligned} C_{bc_{dep}} &= X_{cjc}\cdot C_{jc_{eff}}\left(1-\frac{V_{bc}}{V_{jc}}\right)^{-m_{jc}} && when \quad V_{bc} < 0 \\ C_{bc_{dep}} &= X_{cjc}\cdot C_{jc_{eff}}\left(1+m_{jc}\cdot\frac{V_{bc}}{V_{jc}}\right) && when \quad V_{bc} \ge 0 \end{aligned} \tag{0.85}$$

where *Xcjc* is the partition parameter specifying the ration of base-collector junction capacitance distribution between internal base-internal collector and external base-internal collector.

If *Xcjc*<1, the external base-internal collector capacitance has to be considered:

For **dcap=1**:

$$
\begin{aligned}
C_{bcx} &= (1-X_{cjc})\cdot C_{jc_{eff}}\left(1-\frac{V_{bcx}}{V_{jc}}\right)^{-m_{jc}} && when \quad V_{bcx} < FC\cdot V_{jc} \\
C_{bcx} &= (1-X_{cjc})\cdot C_{jc_{eff}}\cdot\frac{1-FC(1+m_{jc})+m_{jc}\cdot\dfrac{V_{bcx}}{V_{jc}}}{(1-FC)^{1+m_{jc}}} && when \quad V_{bcx} \ge FC\cdot V_{jc}
\end{aligned}
\tag{0.86}
$$

For **dcap=2**:

$$
\begin{aligned}
C_{bcx} &= (1-X_{cjc})\cdot C_{jc_{eff}}\left(1-\frac{V_{bcx}}{V_{jc}}\right)^{-m_{jc}} && when \quad V_{bcx} < 0 \\
C_{bcx} &= (1-X_{cjc})\cdot C_{jc_{eff}}\left(1+m_{jc}\cdot\frac{V_{bcx}}{V_{jc}}\right) && when \quad V_{bcx} \ge 0
\end{aligned}
\tag{0.87}
$$

where *Vbcx* is the voltage between the external base node and the internal collector node.

Substrate Capacitance

The definition of substrate capacitance varies with BJT geometry. For vertical transistors (**subs=1**), substrate capacitance is defined for the base to substrate diode. For lateral transistors (**subs=–1**), it is defined as the capacitance of the substrate to collector diode.

For vertical BJTs,

$$
\begin{aligned}
C_{sc} &= C_{js_{eff}}\left(1-\frac{V_{sc}}{V_{js}}\right)^{-m_{js}} && when \quad V_{sc} < 0 \\
C_{sc} &= C_{js_{eff}}\left(1+m_{js}\cdot\frac{V_{sc}}{V_{js}}\right) && when \quad V_{sc} \ge 0
\end{aligned}
\tag{0.88}
$$

For lateral BJTs,

$$
\begin{aligned}
C_{bs} &= C_{js_{eff}}\left(1-\frac{V_{bs}}{V_{js}}\right)^{-m_{js}} && when \quad V_{bs} < 0 \\
C_{bs} &= C_{js_{eff}}\left(1+m_{js}\cdot\frac{V_{bs}}{V_{js}}\right) && when \quad V_{bs} \ge 0
\end{aligned}
\tag{0.89}
$$

Temperature Dependence

Below is a list of parameters that T-Spice modifies when one or more corresponding temperature coefficients are specified, regardless of **TLEV** values. When neither first nor second order coefficients are specified for a given parameter, the **TLEV**-dependent equations in Saturation Current and Beta on page 337 take precedence.

Forward and reverse current gain (0.90)

$$\beta_f(T) = \beta_f \cdot (1 + TBF_1 \cdot \Delta T + TBF_2 \cdot \Delta T^2)$$
$$\beta_r(T) = \beta_r \cdot (1 + TBR_1 \cdot \Delta T + TBR_2 \cdot \Delta T^2)$$

Voltage and current parameters (0.91)

$$V_{af}(T) = V_{af} \cdot (1 + TVAF_1 \cdot \Delta T + TVAF_2 \cdot \Delta T^2)$$
$$V_{ar}(T) = V_{ar} \cdot (1 + TVAR_1 \cdot \Delta T + TVAR_2 \cdot \Delta T^2)$$
$$I_{tf}(T) = I_{tf} \cdot (1 + TITF_1 \cdot \Delta T + TITF_2 \cdot \Delta T^2)$$

Transit time parameters (0.92)

$$\tau_f(T) = \tau_f \cdot (1 + TTF_1 \cdot \Delta T + TTF_2 \cdot \Delta T^2)$$
$$\tau_r(T) = \tau_r \cdot (1 + TTR_1 \cdot \Delta T + TTR_2 \cdot \Delta T^2)$$

Emission coefficients (0.93)

$$\eta_f(T) = \eta_f \cdot (1 + TNF_1 \cdot \Delta T + TNF_2 \cdot \Delta T^2)$$
$$\eta_r(T) = \eta_r \cdot (1 + TNR_1 \cdot \Delta T + TNR_2 \cdot \Delta T^2)$$
$$\eta_e(T) = \eta_e \cdot (1 + TNE_1 \cdot \Delta T + TNE_2 \cdot \Delta T^2)$$
$$\eta_c(T) = \eta_c \cdot (1 + TNC_1 \cdot \Delta T + TNC_2 \cdot \Delta T^2)$$
$$\eta_s(T) = \eta_s \cdot (1 + TNS_1 \cdot \Delta T + TNS_2 \cdot \Delta T^2)$$

Junction exponents (0.94)

$$\begin{aligned} m_{je}(T) &= m_{je} \cdot (1 + TMJE_1 \cdot \Delta T + TMJE_2 \cdot \Delta T^2) \\ m_{jc}(T) &= m_{jc} \cdot (1 + TMJC_1 \cdot \Delta T + TMJC_2 \cdot \Delta T^2) \\ m_{js}(T) &= m_{js} \cdot (1 + TMJS_1 \cdot \Delta T + TMJS_2 \cdot \Delta T^2) \end{aligned}$$

Parasitic resistor parameters (0.95)

$$\begin{aligned} r_e(T) &= r_e \cdot (1 + TRE_1 \cdot \Delta T + TRE_2 \cdot \Delta T^2) \\ r_b(T) &= r_b \cdot (1 + TRB_1 \cdot \Delta T + TRB_2 \cdot \Delta T^2) \\ r_{bm}(T) &= r_{bm} \cdot (1 + TRM_1 \cdot \Delta T + TRM_2 \cdot \Delta T^2) \\ r_c(T) &= r_c \cdot (1 + TRC_1 \cdot \Delta T + TRC_2 \cdot \Delta T^2) \end{aligned}$$

Energy Gap

The calculation of energy gap is dependent on **TLEV**. For **TLEV=0**, **1**, or **3**, energy gap is always calculated as follows:

$$E_g(T_{nom}) = 1.16 - (7.02 \times 10^4)\frac{T_{nom}^2}{T_{nom} + 1108.0}. \tag{0.96}$$

If **TLEV=2**, the energy gap is calculated as a function of model parameters $E_g(0)$, $GAP1$, and $GAP2$:

$$E_g(T_{nom}) = E_g(0) - GAP1 \cdot \frac{T_{nom}^2}{T_{nom} + GAP2}. \tag{0.97}$$

Saturation Current and Beta

The following equations show the temperature effects for transport saturation current, base-emitter reverse saturation current, and base-collector reverse saturation current.

	TLEV=0, 1, or 2	***TLEV=3***
$I_s(T)$ =	$I_s \cdot e^{facLN}$	$I_s^{(1 + TIS_1 \Delta T + TIS_2 \Delta T^2)}$
$I_{be}(T)$ =	$I_{be} \cdot e^{\frac{facLN}{\eta_f}}$	$I_{be}^{(1 + TIS_1 \Delta T + TIS_2 \Delta T^2)}$

	TLEV=0, 1, or 2	***TLEV=3***
I_{bc}(T) =	$I_{bc} \cdot e^{\frac{facLN}{\eta_r}}$	$I_{bc}^{(1 + TIS_1 \Delta T + TIS_2 \Delta T^2)}$
I_{kf}(T) =	$I_{kf} \cdot (1 + TIKF_1 \Delta T + TIKF_2 \Delta T^2)$	$I_{kf}^{(1 + TIKF_1 \Delta T + TIKF_2 \Delta T^2)}$
I_{kr}(T) =	$I_{kr} \cdot (1 + TIKR_1 \Delta T + TIKR_2 \Delta T^2)$	$I_{kr}^{(1 + TIKR_1 \Delta T + TIKR_2 \Delta T^2)}$
I_{rb}(T) =	$I_{rb} \cdot (1 + TIRB_1 \Delta T + TIRB_2 \Delta T^2)$	$I_{rb}^{(1 + TIRB_1 \Delta T + TIRB_2 \Delta T^2)}$

Temperature-effect equations for leakage saturation currents, bulk-to-collector (or base) saturation current, and beta parameters are shown below.

	TLEV=0 or 2	***TLEV=1***	***TLEV=3***
I_{se}(T) =	$\frac{I_{se}}{\left(\frac{T}{T_{nom}}\right)^{X_{tb}}} \cdot e^{\frac{facLN}{\eta_e}}$	$\frac{I_{se}}{1 + X_{tb}\Delta T} \cdot e^{\frac{facLN}{\eta_e}}$	$I_{se}^{(1 + TISE_1 \Delta T + TISE_2 \Delta T^2)}$
I_{sc}(T) =	$\frac{I_{sc}}{\left(\frac{T}{T_{nom}}\right)^{X_{tb}}} \cdot e^{\frac{facLN}{\eta_c}}$	$\frac{I_{sc}}{1 + X_{tb}\Delta T} \cdot e^{\frac{facLN}{\eta_c}}$	$I_{sc}^{(1 + TISC_1 \Delta T + TISC_2 \Delta T^2)}$
I_{ss}(T) =	$\frac{I_{ss}}{\left(\frac{T}{T_{nom}}\right)^{X_{tb}}} \cdot e^{\frac{facLN}{\eta_s}}$	$\frac{I_{ss}}{1 + X_{tb}\Delta T} \cdot e^{\frac{facLN}{\eta_s}}$	$I_{ss}^{(1 + TISS_1 \Delta T + TISS_2 \Delta T^2)}$
β_f(T) =	$\beta_f \cdot \left(\frac{T}{T_{nom}}\right)^{X_{tb}}$	$\beta_f \cdot (1 + X_{tb}\Delta T)$	$\beta_f \cdot \left(\frac{T}{T_{nom}}\right)^{X_{tb}}$
		(if TBF_1, TBF_2 are not specified)	
β_r(T) =	$\beta_r \cdot \left(\frac{T}{T_{nom}}\right)^{X_{tb}}$	$\beta_r \cdot (1 + X_{tb}\Delta T)$	$\beta_r \cdot \left(\frac{T}{T_{nom}}\right)^{X_{tb}}$
		(if TBR_1, TBR_2 are not specified)	

For *TLEV*=0 and 1

$$facln = \frac{E_g(0)}{V_t(T_{nom})} - \frac{E_g(0)}{Vt(T)} + XTI \cdot ln\left(\frac{T}{T_{nom}}\right) \tag{0.98}$$

For *TLEV*=2

$$facln = \frac{E_g(T_{nom})}{V_t(T_{nom})} - \frac{E_g(T)}{Vt(T)} + XTI \cdot ln\left(\frac{T}{T_{nom}}\right) \tag{0.99}$$

For *TLEV*=3

$$facln = \frac{V_{SB}}{V_t(T_{nom})} - \frac{V_{SB}}{Vt(T)} + XTI \cdot ln\left(\frac{T}{T_{nom}}\right) \tag{0.100}$$

Note: If TBF_1 or TBF_2 is specified, then β_f will be calculated using the equation for **tlev=3**, regardless of the actual value of **tlev**. Similarly, the **tlev=3** equation for β_r will always take precedence if either TBR_1 or TBR_2 is specified.

Capacitance

Capacitance equations are selected by the parameter **TLEVC**.

The following capacitance equations are used when **TLEVC=0**:

$$\begin{aligned} C_{je}(T) &= C_{je} \cdot \left\{1 + m_{je} \cdot \left(4.0 \times 10^{-4} \cdot \Delta T - \frac{V_{je}(T)}{V_{je}} + 1\right)\right\} \\ C_{jc}(T) &= C_{jc} \cdot \left\{1 + m_{jc} \cdot \left(4.0 \times 10^{-4} \cdot \Delta T - \frac{V_{jc}(T)}{V_{jc}} + 1\right)\right\}, \\ C_{js}(T) &= C_{js} \cdot \left\{1 + m_{js} \cdot \left(4.0 \times 10^{-4} \cdot \Delta T - \frac{V_{js}(T)}{V_{js}} + 1\right)\right\} \end{aligned} \tag{0.101}$$

where contact voltages are determined by

$$\begin{aligned} V_{je}(T) &= V_{je} \cdot \left(\frac{T}{T_{nom}}\right) - V_t(T) \cdot \left\{3\ln\left(\frac{T}{T_{nom}}\right) + \frac{E_g(T_{nom})}{V_t(T_{nom})} - \frac{E_g(T)}{V_t(T)}\right\} \\ V_{jc}(T) &= V_{jc} \cdot \left(\frac{T}{T_{nom}}\right) - V_t(T) \cdot \left\{3\ln\left(\frac{T}{T_{nom}}\right) + \frac{E_g(T_{nom})}{V_t(T_{nom})} - \frac{E_g(T)}{V_t(T)}\right\} \\ V_{js}(T) &= V_{js} \cdot \left(\frac{T}{T_{nom}}\right) - V_t(T) \cdot \left\{3\ln\left(\frac{T}{T_{nom}}\right) + \frac{E_g(T_{nom})}{V_t(T_{nom})} - \frac{E_g(T)}{V_t(T)}\right\} \end{aligned} \tag{0.102}$$

If **TLEVC=1**,

$$\begin{aligned} C_{je}(T) &= C_{je} \cdot (1 + C_{te} \cdot \Delta T) \\ C_{jc}(T) &= C_{jc} \cdot (1 + C_{tc} \cdot \Delta T) \\ C_{js}(T) &= C_{js} \cdot (1 + C_{ts} \cdot \Delta T) \end{aligned} \tag{0.103}$$

and contact voltages are determined by

$$\begin{aligned} V_{je}(T) &= V_{je} - T_{vje} \cdot \Delta T \\ V_{jc}(T) &= V_{jc} - T_{vjc} \cdot \Delta T \\ V_{js}(T) &= V_{js} - T_{vjs} \cdot \Delta T \end{aligned} \tag{0.104}$$

If **TLEVC=2,**

$$
\begin{aligned}
C_{je}(T) &= C_{je} \cdot \left(\frac{V_{je}}{V_{je}(T)}\right)^{m_{je}} \\
C_{jc}(T) &= C_{jc} \cdot \left(\frac{V_{jc}}{V_{jc}(T)}\right)^{m_{jc}} \\
C_{js}(T) &= C_{js} \cdot \left(\frac{V_{js}}{V_{js}(T)}\right)^{m_{js}}
\end{aligned} \qquad (0.105)
$$

where

$$
\begin{aligned}
V_{je}(T) &= V_{je} - T_{vje} \cdot \Delta T \\
V_{jc}(T) &= V_{jc} - T_{vjc} \cdot \Delta T \\
V_{js}(T) &= V_{js} - T_{vjs} \cdot \Delta T
\end{aligned} \qquad (0.106)
$$

Note: Use *mje*, *mjc*, and *mjs* instead of *mje*(*T*), *mjc*(*T*), and *mjs*(*T*) in the above equations.

When **TLEVC=3**:

$$
\begin{aligned}
C_{je}(T) &= C_{je} \cdot \left(1 - 0.5 \cdot dvjedt \cdot \frac{\Delta T}{V_{je}}\right) \\
C_{jc}(T) &= C_{jc} \cdot \left(1 - 0.5 \cdot dvjcdt \cdot \frac{\Delta T}{V_{jc}}\right) \\
C_{js}(T) &= C_{js} \cdot \left(1 - 0.5 \cdot dvjsdt \cdot \frac{\Delta T}{V_{js}}\right)
\end{aligned} \qquad (0.107)
$$

and

$$
\begin{aligned}
V_{je}(T) &= V_{je} + dvjedt \cdot \Delta T \\
V_{jc}(T) &= V_{jc} + dvjcdt \cdot \Delta T \\
V_{js}(T) &= V_{js} + dvjsdt \cdot \Delta T
\end{aligned} \qquad (0.108)
$$

where

$$
\begin{aligned}
dvjedt &= -\frac{E_g(T_{nom}) + 3 \cdot V_t(T_{nom}) + [E_g(0) - E_g(T_{nom})] \cdot \left(2 - \frac{T_{nom}}{T_{nom} + GAP2}\right) - V_{je}}{T_{nom}} \\
dvjcdt &= -\frac{E_g(T_{nom}) + 3 \cdot V_t(T_{nom}) + [E_g(0) - E_g(T_{nom})] \cdot \left(2 - \frac{T_{nom}}{T_{nom} + GAP2}\right) - V_{jc}}{T_{nom}} \\
dvjsdt &= -\frac{E_g(T_{nom}) + 3 \cdot V_t(T_{nom}) + [E_g(0) - E_g(T_{nom})] \cdot \left(2 - \frac{T_{nom}}{T_{nom} + GAP2}\right) - V_{js}}{T_{nom}}
\end{aligned} \qquad (0.109)
$$

BJT Level 6 (Mextram)

The level 6 bipolar junction transistor model is the Philips Mextram model.

Parameters

The Mextram model uses the following syntax.

```
.model name npn|pnp level=[6|503|504] | model=modelname [parameters]
```

The Mextram model is fully described in the document **Bipolar Transistor Level 504.**

For additional detailed information about the Mextram model, please refer to the Philips Compact Model Webpage:

http://www.semiconductors.philips.com/Philips_Models/bipolar/mextram

T-Spice includes support for Mextram versions 503 and 504. The version 504 model provides optional support for modeling the extrinsic substrate and the self-heating effects. Selection of an appropriate model is accomplished by using the **model=*modelname*** parameter. If no specific model is specified via the **model** parameter, the **bjt503** model will be selected when level=503 or version=503, otherwise the **bjt504** model will be used.

The available *modelname* values for the Mextram model selection are:

Modelname	***Description***
bjt503	Mextram version 503 with substrate
bjtd503	Mextram version 503 without substrate
bjt504 (default)	Mextram version 504 with substrate
bjtd504	Mextram version 504 without substrate
bjt504t	Mextram version 504 with substrate, self-heating
bjtd504t	Mextram version 504 without substrate, self-heating

BJT Level 9 (VBIC)

In addition to the standard Gummel-Poon BJT model, T-Spice supports the Vertical Bipolar Inter-Company model (VBIC) as the level 9 BJT.

Note: If you specify MOSFET Level 4 in T-Spice it will use the VBIC 1999 version 1.2 model.

Parameters

The VBIC model uses the following syntax.

```
.model name npn|pnp level=9 [parameters]
```

The T-Spice VBIC model is based upon release 1.2 of the VBIC code, and includes certain enhancements for improved convergence and stability.

The following tables describe all of the VBIC BJT parameter.

VBIC Model Parameters

Parameter (alias)	*Description*	*Default*	*Units*
afn	base-emitter flicker noise exponent	1.0	
ajc	base-collector capacitance smoothing factor	-.5	
aje	base-emitter capacitance smoothing factor	-.5	
ajs	substrate-collector capacitance smoothing factor	-.5	
art	base-collector reach-through limiting voltage (0 means infinity)	0.1	V
avc1	base-collector weak avalanche parameter 1	0.0	1/V
avc2	base-collector weak avalanche parameter 2	0.0	1/V
bfn	base-emitter flicker noise 1/f dependence	1.0	
cbco (cbc0)	extrinsic base-collector overlap capacitance	0.0	C
cbeo (cbe0)	extrinsic base-emitter overlap capacitance	0.0	C
ccso	Fixed collector-substrate capacitance	0	C
cjc	base-collector intrinsic zero bias capacitance	0.0	C
cjcp	substrate-collector zero bias capacitance	0.0	C
cje	base-emitter zero bias capacitance	0.0	C
cjep	base-collector extrinsic zero bias capacitance	0.0	C
cth	thermal capacitance	0.0	A
dear	Activation energy shift for ISRR (HBTs)	0	V

VBIC Model Parameters

Parameter (alias)	*Description*	*Default*	*Units*
dtemp (dtmp)	local temperature rise	0	deg C
ea	activation energy for IS	1.12	eV
eaic	activation energy for IBCI/IBEIP	1.12	eV
eaie	activation energy for IBEI	1.12	eV
eais	activation energy for IBCIP	1.12	eV
eanc	activation energy for IBCN/IBENP	1.12	eV
eane	activation energy for IBEN	1.12	eV
eans	activation energy for IBCNP	1.12	eV
eap	Activation energy for ISP	1.12	V
fc	forward bias depletion capacitance limit	0.9	
gamm (gamma)	epi doping parameter	0.0	
hrcf	high current RC factor	0.0	
ibbe	Base-Emitter breakdown current	1e-6	
ibci	ideal base-collector saturation current	1.0E-16	A
ibcip	ideal parasitic base-collector saturation current	0.0	A
ibcn	non-ideal base-collector saturation current	0.0	A
ibcnp	non-ideal parasitic base-collector saturation current	0.0	A
ibei	ideal base-emitter saturation current	1.0E-18	A
ibeip	ideal parasitic base-emitter saturation current	0.0	A
iben	non-ideal base-emitter saturation current	0.0	
ibenp	non-ideal parasitic base-emitter saturation current	0.0	A
ikf	forward knee current (zero means infinity)	0.0	A
ikp	parasitic knee current (zero means infinity)	0.0	A
ikr	reverse knee current (zero means infinity)	0.0	A
is	transport saturation current	1.0E-16	Amps
isp	parasitic transport saturation current	0.0	A
isrr	Reverse saturation current factor (HBTs)	1.0	
itf	coefficient of TF dependence in Ic	0.0	
kfn	base-emitter flicker noise constant	0.0	
mc	base-collector grading coefficient	0.33	
me	base-emitter grading coefficient	0.33	

VBIC Model Parameters

Parameter (alias)	*Description*	*Default*	*Units*
ms	substrate-collector grading coefficient	0.33	
nbbe	Base-Emitter breakdown emission coefficient	1.0	
nci	ideal base-collector emission coefficient	1.0	
ncip	ideal parasitic base-collector emission coefficient	1.0	A
ncn	non-ideal base-collector emission coefficient	2.0	
ncnp	non-ideal parasitic base-collector emission coefficient	2.0	
nei	ideal base-emitter emission coefficient	1.0	
nen	non-ideal base-emitter emission coefficient	2.0	
nf	forward emission coefficient	1.0	
nfp	parasitic fwd emission coefficient	1.0	
nkf	High current roll-off coefficient	0.5	
nr	reverse emission coefficient	1.0	
pc	base-collector built-in potential	0.75	V
pe	base-emitter built-in potential	0.75	V
ps	substrate-collector built-in potential	0.75	V
qbm	Selector for SGP qb formulation	0	
qco (qc0)	epi charge parameter	0.0	C
qtf	variation of TF with base-width modulation	0.0	s
rbi	intrinsic base resistance	0.0	Ω
rbp	parasitic base resistance	0.0	Ω
rbx	extrinsic base resistance	0.0	Ω
rci	intrinsic collector resistance	0.0	Ω
rcx	extrinsic collector resistance	0.0	Ω
re	emitter resistance	0.0	Ω
rs	substrate resistance	0.0	Ω
rth	thermal resistance	0.0	Ω
tavc	temperature coefficient of AVC2	0.0	
td	forward excess-phase delay time	0.0	s
tf	forward transit time	0.0	s
tnbbe	Temperature coefficient of NBBE	0	
tnf	temperature coefficient of NF	0.0	

VBIC Model Parameters

Parameter (alias)	*Description*	*Default*	*Units*
tnom (tref)	nominal measurement temperature of parameters	global **tnom** (25.0)	deg. C
tr	reverse transit time	0.0	s
tvbbe1	First temperature coefficient of VBBE	0	
tvbbe2	Second temperature coefficient of VBBE	0	
vbbe	Base-Emitter breakdown voltage (zero means infinity)	0	V
vef	forward Early voltage (zero means infinity)	0.0	V
ver	reverse Early voltage (zero means infinity)	0.0	V
vers (version)	Version	1.2	
vo (v0)	epi drift saturation voltage	0.0	V
vrev	Revision	0	
vrt	base-collector reach-through limiting voltage (0 means infinity)	0	V
vtf	coefficient of TF dependence on Vbc	0.0	
wbe	portion of IBEI from Vbei (1-WBE from Vbex)	1.0	
wsp	portion of ICCP from Vbep (1-WSP from Vbci)	1.0	
xii	temperature exponent of IBEI/IBCI/IBEIP/IBCIP	3.0	
xikf	Temperature exponent of IKF	0	
xin	temperature exponent of IBEN/IBCN/IBENP/IBCNP	3.0	
xis	temperature exponent of IS	3.0	
xisr	Temperature exponent of ISRR (HBTs)	0	
xrb (xrbi)	temperature exponent of base resistance	0.0	
xrbp	Temperature exponent of extrinsic resistance RBP	0	
xrbx	Temperature exponent of extrinsic resistance RBX	0	
xrc (xrci)	temperature exponent of collector resistance	0.0	
xrcx	Temperature exponent of extrinsic resistance RCX	0	
xre	temperature exponent of emitter resistance	0.0	
xrs	temperature exponent of substrate resistance	0.0	
xtf	coefficient of TF bias dependence	0.0	
xvo (xv0)	temperature exponent of VO	0.0	

The following VBIC model parameters are T-Spice extensions to the standard model.

level	model selector	4	V
ismin	Minimum permissible value of IS	1e-19	A
ispmin	Minimum permissible value of ISP	1e-19	A
mcmin	Minimum permissible value of MC	.01	
memin	Minimum permissible value of ME	.01	
msmin	Minimum permissible value of MS	.01	
rbpmin	Minimum permissible value of RBP	.001	Ω

Equations

For a thorough description of the VBIC model and equations, please refer to the following website:

http://www.designers-guide.org/VBIC/index

Note: Please keep in mind that the default operating temperature and model reference temperature in T-Spice is 25 degrees Celsius, whereas many simulaters use 27 degrees.
When performing simulation comparisons against these simulators, you may want to set the default reference temperature option, **tnom** (page 265), and the operating temperature, **.temp** (page 135), to 27.

BJT Level 10 (Modella)

The level 10 bipolar junction transistor model is the Philips Modella model, a lateral PNP bipolar model.

Parameters

The Modella model uses the following syntax.

```
.model name pnp level=[10|500] | model=modelname [parameters]
```

The complete user manual for the Modella model is located at **Bipolar PNP Transistor Level 500.**

For additional detailed information about the Modella model, please refer to the Philips Compact Model Webpage:

http://www.semiconductors.philips.com/Philips_Models/bipolar/modella

T-Spice includes support for the Modella model version 500 with and without self-heating effects. When the self-heating model is used, the device statement should include an additional temperature node following the substrate node name.

The available *modelname* values for the Modella model selection are:

Modelname	***Description***
bjt500 (default)	Modella version 500 with substrate
bjt500t	Mextram version 500 with substrate, self-heating

Capacitor

Indicates the capacitance of a planar diffused region from geometric and process information.

Parameters

```
.model modelname c [parameter =X]
```

Parameter	*Symbol*	*Description*	*Value*	*Unit*
Cap	*C0*	Capacitance	0	F
Capsw	*Csw*	Sidewall capacitance	0	F/m
Cox	*Cox*	Bottomwall capacitance	0	F/m^2
Del	δ	Difference between drawn and actual widths or lengths	0	m
Di	κ	Dielectric constant or relative permittivity	0	-
L	*L*	Length of the capacitor	0	m
Shrink	*Sshrink*	Shrink factor	1	-
Tc1	*Tc1*	The first temperature coefficient for capacitance	0	1/deg C
Tc2	*Tc2*	The second temperature coefficient for capacitance	0	1/(deg C)^2
Thick	τ	Insulator thickness	0	m
Tnom | Tref	*tref*	Reference temperature	global **tnom** (25.0)	Deg C
W	*W*	Width of capacitor	0	m

Equations

The statement **.model** (page 85) must be associated with an element statement; see the third syntax example in **Capacitor (c)** (page 144).

$$C = MS_{scale}[1 + (T_{c1}\Delta T) + (T_{c2}\Delta T)^2]C_0 \qquad (0.110)$$

Note: When the calculated capacitance is greater than 0.1 F, T-Spice issues a warning message.

The user supplies the multiplicity factor *M* and the scale factor *Scale* in the element statement. See **Capacitor** (page 348). The user can supply *Tc1*, *Tc2*, and *Dtemp* in the element statement, model statement, or both. In the last case, element values override model values. *C0* will be determined in one of 3 ways, which are listed in order of selection:

i) C0 in the element statement.
ii) C0x in the model statement; *C0* will be computed using equation 0.111.
iii) C0 in the model statement.

When *C0x* is supplied in the model description, T-Spice would then compute *C0* as follows:

$$C_0 = L_{eff}W_{eff}C_{ox} + 2(L_{eff} + W_{eff})C_{sw} \tag{0.111}$$

where

$$L_{eff} = L_{scaled} - 2\Delta_{eff} \tag{0.112}$$

$$W_{eff} = W_{scaled} - 2\Delta_{eff} \tag{0.113}$$

$$\text{If } L \text{ is supplied in the element statement:} L_{scaled} = LS_{shrink}S'_{scale} \tag{0.114}$$

$$\text{If } L \text{ is supplied in the model statement:} L_{scaled} = LS_{shrink}S'_{scalm} \tag{0.115}$$

$$\text{If } W \text{ is supplied in the element statement:} W_{scaled} = WS_{shrink}S'_{scale} \tag{0.116}$$

$$\text{If } W \text{ is supplied in the model statement:} W_{scaled} = wS_{shrink}S'_{scale} \tag{0.117}$$

The user supplies S'_{scale} and S'_{scalm} using **.options scale** or **.options scalm**. For further information, see **.options** (page 95).

$$\Delta = \delta S'_{scalm} \tag{0.118}$$

If τ is supplied, T-Spice would compute *Cox* from

$$\text{If } \kappa \neq 0 \text{:} C_{ox} = \kappa(\varepsilon_0/\tau) \tag{0.119}$$

$$\text{If } \kappa = 0 \text{:} C_{ox} = \varepsilon_{0x}/\tau \tag{0.120}$$

The quantities ε_0, $\varepsilon_0 x$ are 8.8542149e-12 and 3.453148e-11, respectively. After this, T-Spice would compute *C0* using the formulas above. If τ is not supplied, T-Spice sets *C0 = 0*.

Examples

```
V1 a d AC 150 0
R1 a b 10
L1 b c 50
.model       capxx        c
+                                             cox=0.25
+                                             capsw=1/3.0
+                                             del=0.1
+                                             shrink=0.5
C1           c            d      capxx
+                                             scale= 0.02/2.5
                                              l = 2*1.2/27.7
                                              w = 2*2.2/27.7

.options scale = 27.7
.AC LIN 1 0.5/3.14159 0.5/3.14159
.PRINT AC IM(V1) IP(V1)
```

Coupled Transmission Line (Level 1)

The coupled transmission line model employs variable electrical parameters.

Parameters

```
.model name cpl level=1 [[r]={matrix}] [l]={matrix} [c]={matrix}
   [[g]={matrix}]
```

Matrices are entered as follows.

```
[r]={r11, r12, r13, ...
+   {r21, r22, r23, ...
+   {r31, r32, r33, ...
... }
```

Parameter	*Description*	*Default*	*Units*
[r]	Optional per unit length resistance matrix. Must be positive definite.	[Zero matrix]	Ω ⁄ m
[l]	Per unit length inductance matrix. Must be positive definite. Off-diagonal elements must be non-negative. **n** and **p** metric prefixes may be used.	[Zero matrix]	H ⁄m
[c]	Per unit length capacitance matrix. Must be positive definite. Off-diagonal elements must be zero or negative. **n** and **p** metric prefixes may be used.	[Zero matrix]	F ⁄m
[g]	Optional per unit length conductance matrix. Must be positive definite.	[Zero matrix]	S ⁄m

Example

An example using lossless symmetrical coupled lines:

```
.model exCPL CPL level=1
+     [l]={494.6n, 63,3n,
+          63.3n, 494.6n }
+     [c]={62.8p, -4.94p,
+          -4.94p, 62.8p }
```

Diode

There are five types of diode models in T-Spice.

- Level 1 describes the Non-Geometric Junction diode model. It is used to model discrete diode devices such as standard and Zener diodes.
- Level 2 describes the Fowler-Nordheim model that is generally used to characterize the tunneling current flow through thin insulator in nonvolatile memory devices.
- Level 3 describes the Geometric Junction diode model. It is used to model IC-based standard silicone-diffused diodes, Schottky barrier diodes and Zener diodes.
- Level 4 describes the Philips Juncap version 2 diode. Note - level 200 and level 9 are also the Juncap model, with different level numbers provided for compatibility with various simulators.
- Level 500 describes the Philips Advanced diode.

Parameters

The Philips Juncap2 model is fully described in the document **Philips Juncap**.

The Philips Advanced Diode is fully described in the document **Philips Diode 500**.

The following model parameter and equation descriptions pertain to Diode levels 1-3.

```
.model name d [parameters]
```

Model Selectors

Parameter	*Symbol*	*Description*	*Default*	*Unit*
level	*level*	Diode model selector	1	—
dcap	*dcap*	Capacitance model selector	2	—
tlev	*tlev*	Temperature equation selector (*tlev*≡3 for Schottky barrier diode)	0	—
tlevc	*tlevc*	Temperature equation selector for junction capacitance and contact potential (*tlevc*=1 and 2 for Schottky barrier diode)	0	—

Geometric and Scaling Parameters

Parameter	*Symbol*	*Description*	*Default*	*Unit*
area	*area*	Junction area (unitless in level=1)	1.0	m^2
PJ	*PJ*	Junction periphery (unitless in level=1)	0.0	m
M	*M*	Multiplier factor to simulate multiple diode in parallel	1	
SCALE	*SCALE*	Scaling factor for geometric parameters	1	
SCALM	*SCALM*	Scaling factor for model parameters	1	

Parameter	***Symbol***	***Description***	***Default***	***Unit***
SHRINK	*SHRINK*	Shrink factor	1	
L	*L*	Length of diode	0.0	m
W	*W*	Width of diode	0.0	m
XW	*XW*	Masking and etching effects	0.0	m
LM	*LM*	Length of metal capacitor	0.0	m
WM	*WM*	Width of metal capacitor	0.0	m
XM	*XM*	Masking and etching effects for *LM* and *WM*	0.0	m
LP	*LP*	Length of polysilicon capacitor	0.0	m
WP	*WP*	Width of polysilicon capacitor	0.0	m
XP	*XP*	Masking and etching effects for *LP* and *WP*	0.0	m

DC Parameters

Parameter	***Symbol***	***Description***	***Default***	***Unit***
IS|JS	*IS*	Saturation current (per unit area). Use calculated *IS* value for Schottky barrier diode if *IS* is not specified; see Application Notes—Schottky Barrier Diodes on page 363.	1.0×10^{-14}	A
JSW	*JSW*	Side wall saturation current (per unit length)	0.0	A
RS	*RS*	Ohmic resistance	0.0	Ω
N	*N*	Emission coefficient (*N*≡1for Schottky barrier diode)	1.0	
BV|VB|VAR|VRB	*BV*	Breakdown voltage	∞	V
IBV|IB	*IBV*	Current at breakdown voltage	1.0×10^{-3}	A
IK|IKF **|IBF**	*IKF*	Forward knee current (per unit area)	0.0	A
IKB|IKR|IBR	*IKB*	Reverse knee current (per unit area)	0.0	A
EXPLI	*EXPLI*	Current explosion model parameter. The PN junction characteristics above the explosion current area are linear, with the slope at the explosion point.	1.0×10^{-15}	A

Capacitance Parameters

Parameter	***Symbol***	***Description***	***Default***	***Unit***
CJ0|CJ|CJA	*Cj0*	Zero-bias bottom wall junction capacitance (per unit area)	0.0	F

Parameter	***Symbol***	***Description***	***Default***	***Unit***
CJP|CJSW	*Cjp0*	Zero-bias side wall/periphery junction capacitance (per unit length)	0.0	F
FC	*FC*	Coefficient for forward-bias depletion area capacitance	0.5	
FCS	*FCS*	Coefficient for forward-bias depletion periphery capacitance	0.5	
MJ|M|EXA	*mj*	Area junction grading coefficient ($mj \equiv 0.5$ for Schottky barrier diode)	0.5	
MJSW|EXP	*mjsw*	Periphery junction grading coefficient	0.33	
PB|PHI|VJ|PHA	*PB*	Area junction contact potential. Use Schottky barrier height as default *PB* value for Schottky barrier diode if PB is not specified.	0.8	V
PHP|VJSW	*PHP*	Periphery junction contact potential	PB	V
TT	τ	Transit time (minority carrier storage time) ($\tau \equiv 0$ for Schottky barrier diode)	0.0	s
XOI	*XOP*	Thickness of oxide contacted on polysilicon	1.0×10^{4}	Å
XOM	*XOM*	Thickness of oxide contacted on metal	1.0×10^{4}	Å

Noise Parameters

Parameter	***Symbol***	***Description***	***Default***	***Unit***
AF	*AF*	Flick noise exponent	1.0	
KF	*KF*	Flick noise exponent	0.0	

Temperature Parameters

Parameter	Symbol	Description	Default	Unit
TNOM | TREF	*Tnom*	Reference temperature	global **tnom** (25.0)	deg

EG	*Eg(0)*	Energy gap at 0°K
		for **tlev** = **2** :
		for **tlev** = **0**, **1**, **3**:
			
			1.16
			1.11
				eV

VSB	*VSB*	Schottky barrier height (**tlev=3** only)	0.8	V
GAP1	*GAP1*	Coefficient in energy gap temperature equation (Si: 4.73×10^{-4}, Ge: 4.77×10^{-4}, and GaAs: 5.41×10^{-4})	7.02×10^{-4}	eV/deg
GAP2	*GAP2*	Coefficient in energy gap temperature equation (Si: 636, Ge: 235, and GaAs: 204)	1108	deg
TCV	*TCV*	Breakdown voltage temperature coefficient	0.0	1/deg

Parameter	Symbol	Description	Default	Unit
TTT1	$\tau t1$	First-order temperature coefficient for τ	0.0	1/deg
TTT2	$\tau t2$	Second-order temperature coefficient for τ	0.0	1/deg^2
TPB	*TPB*	Temperature coefficient for *PB*	0.0	1/deg
TPHP	*TPHP*	Temperature coefficient for *PHP*	0.0	1/deg
CTA|CTC	*CTA*	Temperature coefficient for *CJ*	0.0	1/deg
CTP	*CTP*	Temperature coefficient for *CJP*	0.0	1/deg
TM1	*TM1*	First order temperature coefficient for *mj* (*TM*≡1 for Schottky barrier diode)	0.0	1/deg
TM2	*TM2*	Second order temperature coefficient for *mjsw* (*TM*≡2 for Schottky barrier diode)	0.0	1/deg^2
TRS	*TRS*	Resistance temperature coefficient	0.0	1/deg
XTI	*XTI*	Saturation current temperature exponent (for Schottky barrier diode)	3.0	

Fowler-Nordheim Model Parameters (level=2)

Parameter	***Symbol***	***Description***	***Default***	***Unit***
EF	*EF*	Forward critical electric field	1.0×10^{8}	V/cm
ER	*ER*	Reverse critical electric field	*EF*	V/cm
JF	*JF*	Forward Fowler-Nordheim current coefficient	1.0×10^{-10}	A/V^2
JR	*JR*	Reverse Fowler-Nordheim current coefficient	*JF*	A/V^2
TOX	τox	Thickness of oxide layer	100.0	Å
L	*L*	Length of diode	0.0	m
W	*W*	Width of diode	0.0	m
XW	*XW*	Masking and etching effects	0.0	m

Current Equations

Level 1 and Level 3

In levels 1 and 3, the diodes are modeled in forward bias, reverse bias, and breakdown regions.

In forward and reverse bias regions: *Vd > -BVeff*,

$$I_d = I_{seff} \cdot \left(\exp\left(\frac{V_d}{N \cdot V_t} - 1\right)\right) \tag{0.121}$$

where where *Vt* is thermal voltage, *Vt = kTnom/q*, *Vd* is the voltage across the diode, *Vd = Vnode1-Vnode2*, *BVeff* is the adjusted breakdown voltage, and *N* is emission coefficient (*N*≡1 for Schottky barrier diode).

BVeff can be determined by

$$BV_{eff} = BV - (N \cdot V_t \cdot In)\left(\frac{IBV_{eff}}{I_{break}}\right) \tag{0.122}$$

when *IBVeff>Ibreak*.

and

$$IBVeff = BV, \text{ when } IBVeff \le Ibreak. \tag{0.123}$$

where

$$I_{break} = -I_{seff} \cdot \left(\exp\left(\frac{BV}{N \cdot V_t} - 1\right)\right) \tag{0.124}$$

In breakdown region: *BV <– BVeff*

$$I_d = I_{seff} \cdot \left(\exp\left(\frac{V_d + BV_{eff}}{N \cdot V_t}\right)\right) \tag{0.125}$$

In all the above equations we assume that the diode has a finite breakdown voltage, that is *BV* ≠∞. When BV is not given, or the diode has an infinite breakdown voltage,

$$I_d = I_{seff} \cdot \left(\exp\left(\frac{V_d}{N \cdot V_t} - 1\right)\right) \tag{0.126}$$

Vd>0 is the forward-bias region and *Vd*<0 is the reverse-bias region. There is no breakdown region in this case.

If the high-level injection effects are considered, the current equations are

in forward-bias region (*Vd*>0):

$$I^*_d = \frac{I_d}{1 + \left(\frac{I_d}{I_{KF}}\right)^{\frac{1}{2}}} \tag{0.127}$$

when *IKF*≠0.

$$I^*D=ID \tag{0.128}$$

when *IKF*=0.

and in reverse-bias region ($Vd < 0$)

$$I^*{}_d = \frac{I_d}{1 + \left(\frac{I_d}{I_{KR}}\right)^{\frac{1}{2}}} \tag{0.129}$$

when $IKR \neq 0$.

$$I^*D = ID \tag{0.130}$$

when $IKR=0$.

Diode Capacitance Equations

The diode capacitance *CD* consists of the contributions from diffusion capacitance *Cd*, junction capacitance (depletion capacitance) *Cj*, metal (contact electrode) capacitance *Cm*, and polysilicon (contact electrode) capacitance *Cp*.

Diffusion Capacitance

$$C_d = \pi \cdot \frac{\partial I_d}{\partial V_d} \tag{0.131}$$

Note: For Schottky barrier diodes, $Cd=0$ ($\pi=0$) because the Schottky barrier diode is a majority carrier device and minority carrier effect can be ignored.

Junction Capacitance (Depletion Capacitance)

The junction capacitance has two components: the junction bottom area capacitance *Cja* and the junction periphery capacitance *Cjp*.

$Cj = Cja + Cjp$.

There are two sets of junction capacitance equations selected by the parameter **dcap**.

For **dcap=1**, the junction bottom area capacitance is given by

$$C_{ja} = C_{j0eff} \cdot \left(1 - \frac{V_d}{PB}\right)^{-m_j} \tag{0.132}$$

when $Vd > FC \times PB$

$$C_{ja} = C_{j0eff} \cdot \frac{1 - FC \times (1 + m_j) + m_j \times \frac{V_d}{PB}}{(1 - FC)^{1 + m_j}} \tag{0.133}$$

when $Vd > FC \times PB$

The junction periphery capacitance is given by

$$C_{jp} = C_{jp0eff} \cdot \left(1 - \frac{V_d}{PHP}\right)^{-m_{jsw}} \qquad (0.134)$$

when *Vd* > *FCS* × *PHP*

$$C_{ja} = C_{j0eff} \cdot \frac{1 - FCS \times (1 + m_{jsw}) + m_{jsw} \times \dfrac{V_d}{PB}}{(1 - FCS)^{1 + m_{jsw}}} \qquad (0.135)$$

when *Vd* > *FCS* × *PHP*

The total junction capacitance is

$$C_j = C_{ja} + C_{jp} \qquad (0.136)$$

For **dcap=2** (default), the total junction capacitance is given by

$$C_j = C_{j0eff} \cdot \left(1 - \frac{V_d}{PB}\right)^{-m_j} + C_{jp0eff} \cdot \left(1 - \frac{V_d}{PHP}\right)^{-m_{jsw}} \qquad (0.137)$$

when *Vd* < 0.

$$C_j = C_{j0eff} \cdot \left(1 + m_j \cdot \frac{V_d}{PB}\right) + C_{jp0eff} \cdot \left(1 + m_{sjw} \cdot \frac{V_d}{PHP}\right) \qquad (0.138)$$

when *Vd* > 0.

Metal (Contact Electrode) Capacitance

For **level=3** only

$$C_m = \frac{\varepsilon_{ox}}{X_{OM}} \cdot (W_{Meff} + X_{Meff}) \cdot (L_{Meff} + X_{Meff}) \qquad (0.139)$$

Polysilicon (Contact Electrode) Capacitance

For **level=3** only

$$C_p = \frac{\varepsilon_{ox}}{X_{OP}} \cdot (W_{Peff} + X_{Peff}) \cdot (L_{Peff} + X_{Peff}) \qquad (0.140)$$

Geometric Scaling Effect

Level 1

Scaling for **level=1** involves the use of the junction area (area), junction periphery (*PJ*), and the dimensionless multiplier factor (*M*) to simulate multiple diodes.

Geometric parameters include

$$area_{eff} = area \cdot M \tag{0.141}$$

$$PJ_{eff} = PJ \cdot M \tag{0.142}$$

Element and model parameters include:

$$I_{KEeff} = I_{KF} \cdot area_{eff} \tag{0.143}$$

$$I_{KREeff} = I_{KR} \cdot area_{eff} \tag{0.144}$$

$$IBV_{eff} = IBV \cdot area_{eff} \tag{0.145}$$

$$I_{Seff} = I_S \cdot area_{eff} + JSW \cdot PJ_{eff} \tag{0.146}$$

$$JSW_{eff} = JSW \cdot PJ_{eff} \tag{0.147}$$

$$EXPLI = EXPLI \cdot area_{eff} \tag{0.148}$$

$$C_{j0eff} = C_{j0} \cdot area_{eff} \tag{0.149}$$

$$C_{jp0eff} = C_{jp0} \cdot PJ_{eff} \tag{0.150}$$

$$R_{Seff} = R_s/(area_{eff}) \tag{0.151}$$

Level 3

Level 3 model scaling is affected by the parameters *SCALE*, *SCALM*, *SHRINK*, and *M*.

When both *L* and *W* are specified, geometric parameters include

$$area_{eff} = W_{eff} \cdot L_{eff} \cdot M \tag{0.152}$$

$$PJ_{eff} = (2 \cdot W_{eff} + 2 \cdot L_{eff}) \cdot M \tag{0.153}$$

where

Weff = *W* × *SCALE* × *SHRINK* + *XWeff*

Leff = *L* × *SCALE* × *SHRINK* + *XWeff*

and

XWeff = *XW* × *SCALM*

otherwise

areaeff = *area* × *M* × *SCALE2* × *SHRINK2*

PJeff = PJ × *M* × *SCALE* × *SHRINK*

Geometric parameters for polysilicon and metal capacitance include

LMeff = *LM SCALE* × *SHRINK*

WMeff = *WM SCALE* × *SHRINK*

XMeff = *XM* × *SCALM*

LPeff = *LP SCALE* × *SHRINK*

WPeff = *WP SCALE* × *SHRINK*

XPeff = *XP* × *SCALM*

Element and model parameters include

$$I_{KEeff} = I_{KF} \cdot area_{eff} \qquad (0.154)$$

$$I_{KREeff} = I_{KR} \cdot area_{eff} \qquad (0.155)$$

$$IBV_{eff} = (IBV \cdot area_{eff})/(SCALM^2) \qquad (0.156)$$

$$I_{Seff} = I_S \cdot area_{eff}/SCALM^2 + JSW \cdot PJ_{eff}/SCALM \qquad (0.157)$$

$$JSW_{eff} = (JSW \cdot PJ_{eff})/(SCALM) \qquad (0.158)$$

$$EXPLI = EXPLI \cdot area_{eff} \qquad (0.159)$$

$$R_{Seff} = R_s/(area_{eff} \cdot SCALM^2) \qquad (0.160)$$

$$C_{j0eff} = C_{j0} \cdot area_{eff}/SCALM^2 \qquad (0.161)$$

$$C_{jp0eff} = C_{jp0} \cdot PJ_{eff}/(SCALM) \qquad (0.162)$$

Temperature Effects

Energy Gap

The calculation of energy gap is dependent on **TLEV**. For **TLEV=0**, **1**, or **3**, energy gap is always calculated as follows:

$$E_g(T_{nom}) = 1.16 - (7.02 \times 10^4)\frac{T_{nom}^2}{T_{nom} + 1108.0}\,. \qquad (0.163)$$

If **TLEV=2**, the energy gap is calculated as a function of model parameters $E_g(0)$, $GAP1$, and $GAP2$:

$$E_g(T_{nom}) = E_g(0) - GAP1 \cdot \frac{T_{nom}^2}{T_{nom} + GAP2} . \tag{0.164}$$

Saturation Current

$$I_S(T) = I_S \cdot e^{\frac{facln}{N}} \tag{0.165}$$

$$JSW(T) = JSW \cdot e^{\frac{facln}{N}} \tag{0.166}$$

For *TLEV*=0 and 1

$$facln = \frac{E_g(0)}{V_t(T_{nom})} - \frac{E_g(0)}{Vt(T)} + XTI \cdot ln\left(\frac{T}{T_{nom}}\right) \tag{0.167}$$

For *TLEV*=2

$$facln = \frac{E_g(T_{nom})}{V_t(T_{nom})} - \frac{E_g(T)}{Vt(T)} + XTI \cdot ln\left(\frac{T}{T_{nom}}\right) \tag{0.168}$$

For *TLEV*=3

$$facln = \frac{V_{SB}}{V_t(T_{nom})} - \frac{V_{SB}}{Vt(T)} + XTI \cdot ln\left(\frac{T}{T_{nom}}\right) \tag{0.169}$$

Breakdown Voltage

For *TLEV*=0

$$BV(T) = BV - TCV \cdot \Delta T \tag{0.170}$$

For *TLEV*=1, 2, or 3

$$BV(T) = BV \cdot (1 - TCV \cdot \Delta T) \tag{0.171}$$

where $\Delta T = T - Tnom$

Transit Time

$$\tau(T) = \tau \cdot (1 + \tau_{t1} \cdot \Delta T + \tau_{t2} \cdot \Delta T^2) \tag{0.172}$$

Contact Potential

For *TLEV*=0

$$PB(T) = PB \cdot \left(\frac{T}{T_{nom}}\right) - V_t(T) \cdot \left\{3 \cdot \ln\left(\frac{T}{T_{nom}}\right) + \frac{E_g(T_{nom})}{V_t(T_{nom})} - \frac{E_g(T)}{V_t(T)}\right\} \qquad (0.173)$$

$$PHP(T) = PHP \cdot \left(\frac{T}{T_{nom}}\right) - V_t(T) \cdot \left\{3 \cdot \ln\left(\frac{T}{T_{nom}}\right) + \frac{E_g(T_{nom})}{V_t(T_{nom})} - \frac{E_g(T)}{V_t(T)}\right\} \qquad (0.174)$$

For *TLEV*=1 or 2

$$PB(T) = PB - TPB \times \Delta T \qquad (0.175)$$

$$PHP(T) = PHP - TPHP \times \Delta T \qquad (0.176)$$

For *TLEV*=3

$$PB(T) = PB + dpbdt \times \Delta T \qquad (0.177)$$

$$PHP(T) = PHP - dphpdt \times \Delta T \qquad (0.178)$$

where

TLEV= 2

$$dpbdt = \qquad (0.179)$$

$$-\frac{E_g(T_{nom}) + 3 \cdot V_t(T_{nom}) + [E_g(0) - E_g(T_{nom})] \cdot \left(2 - \frac{T_{nom}}{T_{nom} + GAP2}\right) - PB}{T_{nom}}$$

$$dphpdt = \qquad (0.180)$$

$$-\frac{E_g(T_{nom}) + 3 \cdot V_t(T_{nom}) + [E_g(0) - E_g(T_{nom})] \cdot \left(2 - \frac{T_{nom}}{T_{nom} + GAP2}\right) - PHP}{T_{nom}}$$

and

TLEV=0 or 1

Eg(0) and GAP2 take their default values in the above equations:

$Eg(0) = 1.16$

$GAP2 = 1108.0$

Junction Capacitance

TLEV=0

$$C_j(T) = C_{j0} \cdot \left\{1 + m_j \cdot \left(4.0\times10^{-4} \cdot \Delta T - \frac{PB(T)}{PB} + 1\right)\right\} \tag{0.181}$$

$$C_{jsw}(T) = C_{jsw0} \cdot \left\{1 + m_j \cdot \left(4.0\times10^{-4} \cdot \Delta T - \frac{PHP(T)}{PHP} + 1\right)\right\} \tag{0.182}$$

TLEV=1

$$C_j(T) = C_{j0} \cdot (1 + CTA \cdot \Delta T) \tag{0.183}$$

$$C_{jsw}(T) = C_{jsw0} \cdot (1 + CTP \cdot \Delta T) \tag{0.184}$$

TLEV=2

$$C_j(T) = C_{j0} \cdot \left(\frac{PB}{PB(T)}\right)^{m_j} \tag{0.185}$$

$$C_{jsw}(T) = C_{jsw0} \cdot \left(\frac{PHP}{PHP(T)}\right)^{m_{jsw}} \tag{0.186}$$

Note: Use *mj* instead of *mj*(*T*) in the above equation.

TLEV=3

$$C_j(T) = C_{j0} \cdot 1 - 0.5 \cdot dpbdt \cdot \frac{\Delta T}{PB} \tag{0.187}$$

$$C_{jsw}(T) = C_{jsw0} \cdot \left(1 - 0.5 \cdot dphpdt \cdot \frac{\Delta T}{PHP}\right) \tag{0.188}$$

Grading Coefficient

$$m_j(T) = m_j \cdot (1 + TM1 \cdot \Delta T + TM2 \cdot \Delta T^2) \tag{0.189}$$

Resistance

$$R_S(T) = R_S \cdot (1 + TRS \cdot \Delta T) \tag{0.190}$$

Fowler-Nordheim Model (Level 2)

The Fowler-Nordheim model is used to characterize the tunneling current flow through thin insulators in nonvolatile memory devices such as the floating gate devices and the MIOS (metal-insulator-oxide-semiconductor) devices. The insulators in these devices are sufficiently thin (about 100 Å) to permit tunneling of carriers.

Current Equations

$$I_d = area_{eff} \cdot J_F \cdot \left(\frac{v_d}{t_{ox}}\right)^2 \cdot e^{-\frac{E_F \cdot t_{ox}}{V_d}} \qquad (0.191)$$

when $Vd \geq 0$

$$I_d = -area_{eff} \cdot J_R \cdot \left(\frac{v_d}{t_{ox}}\right)^2 \cdot e^{-\frac{E_R \cdot t_{ox}}{V_d}} \qquad (0.192)$$

when $Vd < 0$

where

$areaeff = Weff \times Leff \times M$

and

$Weff = W \times SCALM \times SHRINK + XWeff$

$Leff = L \times SCALM \times SHRINK + XWeff$

$XWeff = XW \times SCALM$

Capacitance

$$C_D = area_{eff} \cdot \frac{\varepsilon_{ox}}{t_{ox}} \qquad (0.193)$$

Application Notes—Schottky Barrier Diodes

The Schottky barrier diode is not explicitly modeled in T-Spice, but it can be simulated using the PN junction diode model provided extra attention is paid to the differences between the PN junction diode and Schottky barrier diode.

Schottky barrier diodes and PN junction diodes have a similar I-V relation, but their saturation current expressions are quite different. For Schottky barrier diode, the saturation current is given by

$$I_S = A_{RC} \cdot K_{RC} \cdot T_{nom}{}^2 \cdot \exp\left(-\frac{V_{sb}}{V_t}\right) \qquad (0.194)$$

where $A_{RC} = 1.2 \times 10^6 (A \cdot m^{-2} \cdot K^{-2})$ is the Richardson constant, *KRC* is the ratio of the effective Richardson constant to the Richardson constant, *VSB* is the Schottky barrier height, and $V_t = (kT_{nom})/q$ is the thermal voltage. Use this calculated *IS* value for Schottky barrier diode if *IS* is not specified.

Some typical *KRC* values are:

Type	***Si***	***Ge***	***GaAs***
p-type	0.66	0.34	0.62
n-type	▪ (111): 2.2 ▪ (100): 2.1	▪ (111): 1.11 ▪ (100): 1.19	0.068

Some other parameter values intended for Schottky barrier diodes are indicated in the diode parameter list. Use these values instead of default in the simulation of Schottky barrier diode if these parameters are not specified.

JFET

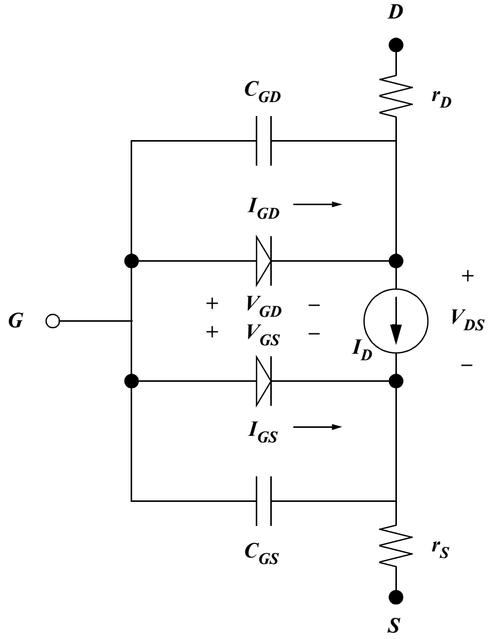
The junction field-effect transistor model uses the basic FET model of Schichmann and Hodges. The DC characteristics are modeled by the threshold voltage and the gain factor, and charge storage is modeled by two reverse-biased PN junctions. Source and drain series resistances are included. JFET models are always level 0.

Parameters

```
.model name njf|pjf [parameters]
```

Parameter	*Symbol*	*Description*	*Default*	*Units*
vto	*Vt0*	Threshold voltage	–2.0	V
beta	β	Transconductance parameter	1.0×10^{-4}	A/V^2
lambda	λ	Channel length modulation parameter	0.0	1/V
rd	*Rd*	Drain series resistance	0.0	Ω
rs	*Rs*	Source series resistance	0.0	Ω
cgs	*Cgs*	Zero-bias gate-source junction capacitance	0.0	F
cgd	*Cgd*	Zero-bias gate-drain junction capacitance	0.0	F
pb	*PB*	Gate junction potential	1.0	V
is	*IS*	Gate junction saturation current	1.0×10^{-14}	A
fc	*FC*	Forward-bias depletion capacitance coefficient	0.5	—

Large-Signal Model



Equations

Currents

In the normal or forward region of operation, the DC currents are described by the following equations, based on the quadratic FET model of Shichmann and Hodges.

For $(VGS - VTO) \leq 0$,

$$I_{DS} = 0 \tag{0.195}$$

For $0 < (VGS - VTO) \leq VDS$,

$$I_{DS} = \beta(V_{GS} - V_{TO})^2(1 + \lambda V_{DS}) \tag{0.196}$$

For $0 < VDS < (VGS - VTO)$,

$$I_{DS} = \beta V_{DS}[2(V_{GS} - V_{TO}) - V_{DS}](1 + \lambda V_{DS}) \tag{0.197}$$

For the inverse or reverse region of operation where $VDS < 0$, the same set of equations is used, with *VGS* replaced by *VGD* and the signs on the *VDS* terms reversed.

For $(VGD - VTO) \leq 0$,

$$I_{DS} = 0 \tag{0.198}$$

For $0 < (VGD - VTO) \leq VDS$,

$$I_{DS} = \beta(V_{GD} - V_{TO})^2(1 - \lambda V_{DS}) \tag{0.199}$$

For $0 < VDS < (VGD - VTO)$,

$$I_{DS} = \beta V_{DS}[2(V_{GD} - V_{TO}) - V_{DS}](1 - \lambda V_{DS}) \tag{0.200}$$

The gate-to-drain and gate-to-source leakage currents are:

$$I_{GD} = I_S \cdot \left(e^{\frac{qV_{GD}}{kT}} - 1\right) \tag{0.201}$$

$$I_{GS} = I_S \cdot \left(e^{\frac{qV_{GS}}{kT}} - 1\right) \tag{0.202}$$

The total currents are then:

$$I_D = I_{DS} - I_{GD} \tag{0.203}$$

$$I_G = I_{GD} + I_{GS} \tag{0.204}$$

$$I_S = -(I_{DS} + I_{GS}) \tag{0.205}$$

Charges

The charge equations corresponding to *CGD* and *CGS* are based on reverse-biased P-N step junctions.

For $VGX < FC \cdot PB$,

$$Q_{GX} = 2C_{GX} \cdot PB\left(1 - \sqrt{1 - \frac{V_{GX}}{PB}}\right) \tag{0.206}$$

For $VGX \geq FC \cdot PB$,

$$Q_{GX} = \frac{C_{GX}\left[\left(1 - \frac{3}{2}FC\right)(V_{GX} - FC \cdot PB) + \frac{1}{4PB}(V_{GX}^2 - (FC \cdot PB)^2)\right]}{(1 - FC)\sqrt{1 - FC}} + C_{GX} \cdot PB \cdot (1 - \sqrt{1 - FC}) \tag{0.207}$$

X denotes either source (*S*) or drain (*D*). The junction charge equations are identical to those used by the BJT and MOSFET models, except that the grading coefficient has been fixed at 0.5.

MESFET

Parameters

```
.model name nmf|pmf |njf|pjf [parameters]
```

For HSPICE compatibility, you can create a MESFET device in T-Spice using a name which begins with the letter j. The syntax for the MESFET device statement is exactly as documented in MESFET (z) on page 163, except that the device name is **jname** instead of **zname**.

Additionally, the MESFET model can use either the **.model** name [**nmf | pmf**] naming convention or **.model** name [**njf | pjf**].

Submodel Selectors

Parameter	*Symbol*	*Description*	*Default*	*Units*
level	LEVEL	Level selector (1, 2 or 3)	1	
sat	*SAT*	Model selector	0	—
capop	*CAPOP*	Capacitance model selector	0	—
dcap	*DCAP*	Forward-biased diode equation selector (*CAPOP*=0)	2	—
tlev	*TLEV*	Temperature model selector	0	—
tlevc	*TLEVC*	Junction capacitance temperature model selector	0	—
nlev	*NLEV*	Channel thermal noise equation selector	2	—

DC Parameters

Parameter	*Symbol*	*Description*	*Default*	*Units*
alpha	α	Saturation voltage factor	2.0	V^{-1}
b	*b*	Doping tail extending parameter	0.3	V^{-1}
beta	β	Transconductance parameter	2.5×10^{-3}	A/V^2
d	*D*	Channel dielectric constant	11.7 (Si)	—
gamma | gamds	γds	Drain-induced *VT0* lowering	0.0	A
is	*Isat*	Gate saturation current	1.0×10^{-14}	A
k1	*k1*	Body effect on *VT0*	0.0	V^{-1}
lambda	λ	Channel length modulation parameter	0.0	V^{-1}
nchan	*ND*	Channel doping concentration	1.552×10^{16}	cm^{-3}
rd	*Rd*	Drain ohmic resistance	0.0	Ω
rg	*Rg*	Gate resistance	0.0	Ω

Parameter	*Symbol*	*Description*	*Default*	*Units*
rs	*Rs*	Source ohmic resistance	0.0	Ω
rsh	*Rsh*	Source/drain sheet resistance	0.0	Ω/m^2
rshg	*Rshg*	Gate sheet resistance	0.0	Ω/m^2
rshl	*Rshl*	Channel sheet resistance	0.0	Ω/m^2
satexp	*nvds*	Drain voltage exponent for **sat**=3	3.0	—
ucrit	*Ec*	Critical mobility degradation field	0.0	V/cm
vbi	*Vbi*	Gate diode built-in voltage	1.0	V
vgexp	*nvgst*	Gate voltage exponent for **sat**=3	2.0	—
vp	*Vp*	Channel pinch-off voltage	Computed.	V
vto	*VT0*	Threshold voltage	Computed.	V

Capacitance Parameters

Parameter	*Symbol*	*Description*	*Default*	*Units*
cgs	*Cgs*	Zero bias G-S capacitance	0.0	F
cgd	*Cgd*	Zero bias G-D capacitance	0.0	F
crat	*CRAT*	Source fraction of *GCAP*	0.666	—
fc	*FC*	Forward bias depletion capacitance coefficient	0.5	—
gcap	*GCAP*	Total zero bias gate capacitance	—	F
interr	*Eint*	Integration error bound (*CAPOP*=1)	0.01	—
m	*m*	Junction grading coefficient (*CAPOP*=0)	0.5	—
pb	$\phi s0$	Gate junction potential	0.8	V
tt	τ	Transit time (*CAPOP*=0)	0.0	s
vdel	δ	Transition width for *Vgs* (*CAPOP*=1)	0.2	V
vmax	*Vmax*	*Vn* limiting value (*CAPOP*=1)	0.5	V

Noise Parameters

Parameter	*Symbol*	*Description*	*Default*	*Units*
af	*Af*	Flicker noise exponent	1.0	—
gdsnoi	*GDSNOI*	Thermal channel noise coefficient (*NLEV*=3)	1.0	—
kf	*Kf*	Flicker noise coefficient	0.0	—

Geometry Parameters

Parameter	*Symbol*	*Description*	*Default*	*Units*
acm	*ACM*	Area calculation method selector	0	—
align	*ALIGN*	Misalignment of gate	0.0	m
hdif	*Hdif*	Space between S/D contacts and junction	0.0	m
l	*L*	Default gate length	0.0	m
ldel	*Ldel*	Delta between drawn and optical gate length	0.0	m
ldif	*Ldif*	Distance from junction to gate edge	0.0	m
w	*W*	Default gate width	0.0	m
wdel	*Wdel*	Delta between drawn and optical gate width	0.0	m

Area Calculation Method (ACM) Parameters

	ACM=0	*ACM=1*
AREAeff	$\frac{Weff}{Leff} \cdot M$	$Weff \cdot Leff \cdot M$
RDeff	$\frac{RD}{AREAeff}$	**If RD≠0:** $\frac{RD}{M}$ **If RD=0:** $\frac{DIF}{ff \cdot M} + RSHL \cdot \frac{LDIF + ALIGN}{Weff \cdot M}$
RSeff	$\frac{RS}{AREAeff}$	**If RS≠0:** $\frac{RS}{M}$ **If RS=0:** $RSH \cdot \frac{HDIF}{Weff \cdot M}$ $+ RSHL \cdot \frac{LDIF - ALIGN}{Weff \cdot M}$
RGeff	$RG \cdot \frac{AREAeff}{M^2}$	**If RG≠0:** $\frac{RG}{M}$ **If RG=0:** $RSHG \cdot \frac{Weff}{Leff \cdot M}$

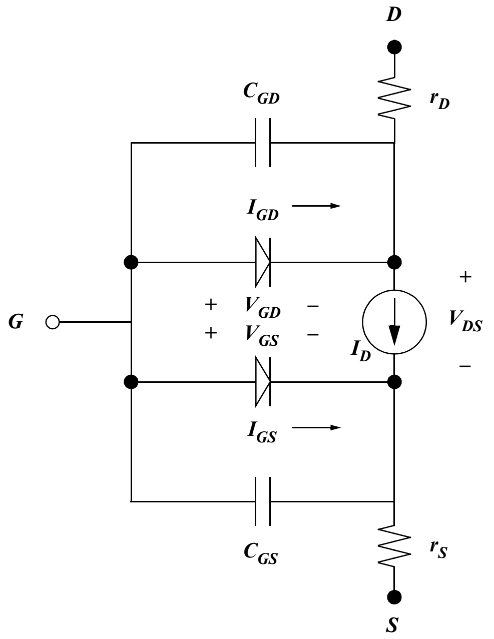
	ACM=0	*ACM=1*
ISeff	$IS \cdot AREAeff$	$IS \cdot AREAeff$
CGSeff	$CGS \cdot AREAeff$	$CGS \cdot AREAeff$
CGDeff	$CGD \cdot AREAeff$	$CGD \cdot AREAeff$
BETAeff	$BETA \cdot \frac{Weff}{Leff} \cdot M$	$BETA \cdot \frac{Weff}{Leff} \cdot M$

Note that the model parameters ***IS***, ***CGS***, and ***CGD*** are unitless when **ACM=0** and per square meter when **ACM=1**. For example, when **ACM=0**, $CGS = 5 \times 10^{-12}$, $CGD = 1.4 \times 10^{-11}$, and $IS = 1 \times 10^{-14}$; for **ACM=1**, however, $CGS = 5$, $CGD = 14$, and $IS = 1 \times 10^{-2}$.

Temperature Dependence Parameters

Parameter	*Symbol*	*Description*	*Default*	*Units*
bex	*BEX*	Mobility temperature exponent	0.0	—
ctd	*CTD*	Temperature coefficient for *Cgd*	0.0	deg^{-1}
cts	*CTS*	Temperature coefficient for *Cgs*	0.0	deg^{-1}
eg	*Eg*	Energy gap for G-D and G-S diodes	1.16	eV
gap1	*GAP1*	First-order bandgap correction	7.02e-4	eV ⁄deg
gap2	*GAP2*	Second-order bandgap correction	1108	deg
n	*N*	G-S and G-D diode emission coefficient	1.0	—
tcv	*TCV*	Temperature coefficient for *VT0*	0.0	deg^{-1}
tpb	*TPB*	Temperature coefficient for *pb*	0.0	deg^{-1}
trd	*TRD*	Temperature coefficient for *Rd*	0.0	deg^{-1}
trg	*TRG*	Temperature coefficient for *Rg*	0.0	deg^{-1}
trs	*TRS*	Temperature coefficient for *Rs*	0.0	deg^{-1}
xti	*XTI*	*Isat* temperature exponent	0.0	—

Large-Signal Model



Equations

There are three levels of the T-Spice MESFET model.

- *Level 1*. The Curtice model—a diode-based capacitance model and a simplified *Ids* calculation.
- *Level 2*. The Statz model (Statz et al. 1987)—a revision of the Curtice model, with an improved capacitance model and a more sophisticated *Ids* calculation.
- *Level 3*. An HSPICE-compatible model—highly customizable.

Currents

For all three levels,

$$I_{ds} = \beta \cdot \frac{(V_{gst})^{VGEXP}}{A} \cdot B \cdot (1 + \lambda V_{ds}) \quad (0.208)$$

where

$$V_{gst} = V_{gs} - V_{T0} - \gamma_{ds} \cdot V_{ds} \quad (0.209)$$

A is the doping profile extension factor. At level 1, *A* = 0; otherwise:

$$A = 1 + bV_{gst} \quad (0.210)$$

VGEXP can be varied only at level 3; at levels 1 and 2, it is fixed at 2.0.

B is the saturation term; its form depends on the value of *SAT*.

When *SAT* = 0:

$$B = \tanh(\alpha V_{ds}) \tag{0.211}$$

This form is fixed (that is, *SAT* can only take a value of 0) at level 1.

When *SAT* = 1:

$$B = \tanh\left(\alpha \cdot \frac{V_{ds}}{V_{gst}}\right) \tag{0.212}$$

When *SAT* = 2:

$$B = 1 - \left(1 - \alpha \cdot \frac{V_{ds}}{3}\right)^3 \tag{0.213}$$

When $Vds > 3/\alpha$, $B = 1$. This form is fixed (that is, *SAT* can only take a value of 2) at level 2.

When $SAT = 3$ and $Vds < SATEXP/\alpha$:

$$B = 1 - \left(1 - \alpha \cdot \frac{V_{ds}}{SATEXP}\right)^{SATEXP} \tag{0.214}$$

Otherwise, *B* = 1.

Capacitances

There are two capacitance models, one at level 1 (a “diode-like” model) and one at level 2 (the Statz model). The *CAPOP* parameter selects between the models.

When *CAPOP* = 0:

When the junctions are reverse-biased ($Vgd < FC \cdot \phi s0$):

$$C_{gd} = C_{gd}(0) \cdot \left(1 - \frac{V_{gd}}{\phi_{s0}}\right)^{-m} \tag{0.215}$$

$$C_{gs} = C_{gs}(0) \cdot \left(1 - \frac{V_{gs}}{\phi_{s0}}\right)^{-m} \tag{0.216}$$

When the junctions are forward-biased, two models are available; the *DCAP* parameter selects between them. When *DCAP* = 1:

$$C_{gd} = \tau \cdot \frac{\partial I_{gd}}{\partial V_{gd}} + C_{gd}(0) \cdot \frac{1 - FC(1+m) + m\dfrac{V_{gd}}{\phi_{s0}}}{(1-FC)^{m+1}} \qquad (0.217)$$

$$C_{gs} = \tau \cdot \frac{\partial I_{gs}}{\partial V_{gs}} + C_{gs}(0) \cdot \frac{1 - FC(1+m) + m\dfrac{V_{gs}}{\phi_{s0}}}{(1-FC)^{m+1}} \qquad (0.218)$$

When *DCAP* = 2 (default):

$$C_{gd} = \tau \cdot \frac{\partial I_{gd}}{\partial V_{gd}} + C_{gd}(0) \cdot \left(1 + m\frac{V_{gd}}{\phi_{s0}}\right) \qquad (0.219)$$

$$C_{gs} = \tau \cdot \frac{\partial I_{gs}}{\partial V_{gs}} + C_{gs}(0) \cdot \left(1 + m\frac{V_{gs}}{\phi_{s0}}\right) \qquad (0.220)$$

When *CAPOP* = 1:

The basic Statz capacitance equations are:

$$\begin{aligned} C_{gs,\,gd} = {} & \frac{1}{4}\frac{C_{gs}(0)}{\sqrt{1 - \dfrac{V_n}{\phi_{s0}}}} \times \left[1 + \frac{V_{eff} - V_{T0}}{\sqrt{(V_{eff} - V_{T0})^2 + \delta^2}}\right]\left[1 \pm \frac{V_{gs} - V_{gd}}{\sqrt{(V_{gs} - V_{gd})^2 + (\alpha)^{-2}}}\right] \\ & + \frac{1}{2} \times C_{gd}(0)\left[1 \mp \frac{V_{gs} - V_{gd}}{\sqrt{(V_{gs} - V_{gd})^2 + (\alpha)^{-2}}}\right] \end{aligned} \qquad (0.221)$$

where

$$V_{eff} = \frac{1}{2}[V_{gs} + V_{gd} + \sqrt{(V_{gs} - V_{gd})^2 + (\alpha)^{-2}}] \qquad (0.222)$$

$$V_n = \frac{1}{2}[V_{eff} + V_{T0} + \sqrt{(V_{eff} - V_{T0})^2 + (\delta)^2}] \qquad (0.223)$$

If $V_n > V_{max}$, then *Vn* is limited to *Vmax*. In the plus/minus signs (±) above, the top signs hold for *Cgs,* the bottom ones for *Cgd*.

Temperature Dependence Equations

In the following, $\Delta T = T - T_{ref}$. T_{ref} is the temperature at which the user-supplied parameters are valid, which defaults to 25° C. All temperatures in the following equations are assumed to be in Kelvin.

E_g temperature dependence. For all values of TLEV, the energy gap E_g at the reference and simulation temperatures is calculated using

$$E_g(T) = EG - GAP1 \times \frac{T}{T + GAP2} \tag{0.224}$$

For *TLEV* 0 and 1, *EG*, *GAP1*, and *GAP2* are held fixed at 1.16, 7.02×10^{-4}, and 1108.0, respectively, regardless of what values the user specifies for these parameters.

Saturation current temperature dependence. The saturation current temperature extrapolation is calculated by the following general equation:

$$I_s(T) = IS \cdot \exp\left\{\frac{e}{k_B N}\left(\frac{E_g(T_{ref})}{T_{ref}} - \frac{E_g(T)}{T} + XTI \cdot In\left(\frac{T}{T_{ref}}\right)\right)\right\} \tag{0.225}$$

For TLEV 0 and 1, the user-supplied *EG* is used in place of $E_g(T)$ for all *T*. For TLEV=2, $E_g(T) = E_g(T)$, as calculated above.

Capacitance parameters temperature dependence. A separate selection parameter, TLEVC, is used to choose one of four methods of temperature-adjusting the gate capacitance values. This parameter also influences the temperature compensation of ϕ_{s0}.

For TLEVC=0, the gate junction potential (θ_{s0}) is temperature-adjusted as follows:

$$\phi_{s0}(T) = \Phi_{s0} \cdot \frac{T}{T_{ref}} - \frac{3k_B}{e} T \ln\left(\frac{T}{T_{ref}}\right) - E_g(T_{ref}) \cdot \frac{T}{T_{ref}} + E_g(T) \tag{0.226}$$

The gate capacitances are calculated by

$$C_{gs}(T) = C_{gs} \cdot \left[1 + m \cdot \left(4.0 \times 10^{-4} \cdot \Delta T - \frac{\phi_{s0(T)}}{\phi_{s0(T_{ref})}} + 1\right)\right] \tag{0.227}$$

$$C_{gd}(T) = C_{gs} \cdot \left[1 + m \cdot \left(4.0 \times 10^{-4} \cdot \Delta T - \frac{\phi_{s0(T)}}{\phi_{s0(T_{ref})}} + 1\right)\right] \tag{0.228}$$

For TLEVC=1, the junction potential is temperature-adjusted with

$$\phi_{s0}(T) = \phi_{s0} - TPB \cdot \Delta T \tag{0.229}$$

The gate capacitances are calculated by

$$C_{gs}(T) = C_{gs} \cdot (1 + CTS + \Delta T) \tag{0.230}$$

$$C_{gd}(T) = C_{gd} \cdot (1 + CTS \cdot \Delta T) \tag{0.231}$$

For TLEVC=2, the junction potential is calculated as it is for TLEVC=1. The gate capacitances are calculated by

$$C_{gs}(T) = C_{gs} \cdot \left(\frac{\phi_{s0}}{\phi_{s0^T}}\right)^m \tag{0.232}$$

$$C_{gs}(T) = C_{gd} \cdot \left(\frac{\phi_{s0}}{\phi_{s0^T}}\right)^m \tag{0.233}$$

For TLEVC=3, the junction potential calculation attempts to estimate the term $d\phi_{s0}/(dT)$ (*dpbdt*) using the following equations, depending on TLEV.

For TLEV=0 or 1,

$$dpbdt = -\left(\frac{1}{T_{ref}}E_g(T_{ref}) + \frac{3k_B}{e}T_{ref} + (1.16 - E_g(T_{ref})) \cdot \left(2 - \frac{T_{ref}}{T_{ref} + 1108}\right)(-\phi_{s0})\right) \tag{0.234}$$

For TLEV=2,

$$dpbdt = -\left(\frac{1}{T_{ref}}E_g(T_{ref}) + \frac{3k_B}{e}T_{ref} + (EG - E_g(T_{ref})) \cdot \left(2 - \frac{T_{ref}}{T_{ref} + GAP2}\right)(-\phi_{s0})\right) \tag{0.235}$$

The gate capacitances are then calculated by

$$C_{gs}(T) = C_{gs} \cdot \left(1 - 0.5 \cdot dpbdt \times \frac{\Delta T}{\phi_{s0}}\right) \tag{0.236}$$

$$C_{gc}(T) = C_{gd} \cdot \left(1 - 0.5 \cdot dpbdt \times \frac{\Delta T}{\phi_{s0}}\right) \tag{0.237}$$

Note that $C_{gs,\,gd}$ becomes what is referred to elsewhere as $C_{gs,\,gd}(0)$, the zero-bias gate capacitances.

V_{TO} temperature dependence. The threshold voltage is temperature-compensated with a simple linear model

$$V_{TO}(T) = V_{TO} - TCV \times \Delta T \tag{0.238}$$

Transconductance temperature dependence. The temperature compensation of the transconductance factor (β) is calculated as follows

$$\beta(T) = \beta \cdot \left(\frac{T}{T_{ref}}\right)^{BEX} \tag{0.239}$$

Parasitic resistance temperature dependence. The source, drain, and gate parasitic resistances are temperature-compensated by the following equations

$$R_d(T) = R_d \cdot (1 + TRD \times \Delta T) \tag{0.240}$$

$$R_s(T) = R_s \cdot (1 + TRS \times \Delta T) \tag{0.241}$$

$$R_g(T) = R_g \cdot (1 + TRG \times \Delta T) \tag{0.242}$$

MOSFET Levels 1/2/3 (Berkeley SPICE 2G6)

Parameters

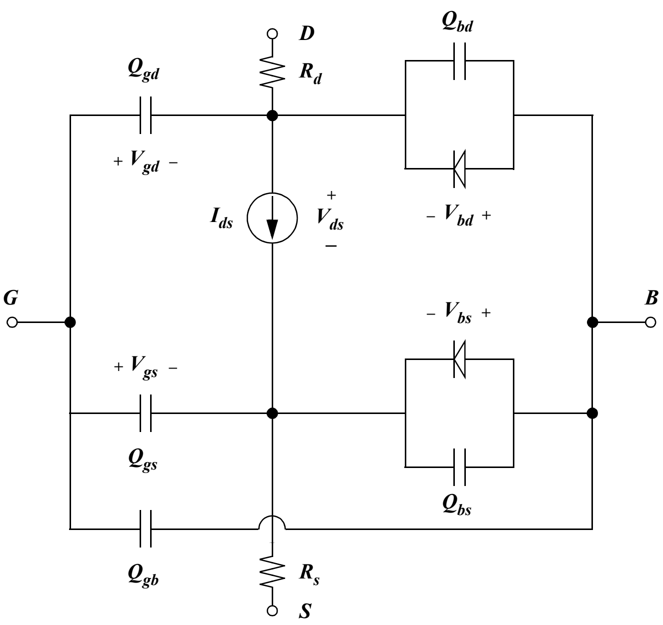
```
.model name nmos|pmos level=1|2|3 [parameters]
```

Also see Additional MOSFET Parameters on page 433.

Parameter	*Symbol*	*Description*	*Default*	*Units*	
level	*LEVEL*	Model selector	1	—	❶❷❸
vto	*Vt0*	Zero-bias threshold voltage	Computed	V	❶❷❸
kp	*KP*	MOSFET transconductance	Computed	A/V^2	❶❷❸
gamma	γ	Bulk threshold parameter	Computed	$V^{1/2}$	❶❷❸
phi	ϕ	Surface potential	Computed	V	❶❷❸
lambda	λ	Channel length modulation	Computed	V^{-1}	❶❷
tox	*Tox*	Gate oxide thickness	Computed.	If < 1, m; if > 1, Å	❶❷❸
nsub | dnb | nb	*Nsub*	Bulk doping concentration	0.0	cm^{-3}	❶❷❸
nss	*NSS*	Surface state density	0.0	cm^{-2}	❷❸
nfs | dnf | nf | dfs	*NFS*	Fast surface state density	Computed	cm^{-2}	❶❷❸
xj	*Xj*	Junction depth	0.0	m	❶❷❸
ld	*LD*	Lateral diffusion	0.75 × *xj*	m	❶❷❸
wd	*WD*	Width diffusion	0.0	m	❶❷❸
xl | dl | ldel	*Xl*	Mask and etching length change	0.0	m	❶❷❸
xw | dw | wdel	*Xw*	Mask and etching width change	0.0	m	❶❷❸
uo	$\mu 0$	Electron or hole mobility	600.0	$cm^2/V \cdot s$	❶❷❸
ucrit	*UCRIT*	Critical field mobility degradation coefficient	1.0×10^{-4}	V/cm	❷
uexp	*UEXP*	Critical field mobility degradation exponent	0.0	—	❷
vmax	*VMAX*	Maximum carrier drift velocity	0.0	m/s	❷❸
neff	*NEFF*	Total channel charge coefficient	1.0	—	❷
delta	δ	Width effect on threshold voltage	0.0	—	❷❸
tpg	*TPG*	Type of gate material:	1	—	❶❷❸
		▪ Doping type same as source-drain (**tpg=1**)			❶❷❸
		▪ Doping type reverse of source-drain (**tpg=-1**)			❶❷❸

Parameter	*Symbol*	*Description*	*Default*	*Units*	
		▪ Aluminum gate (**tpg=0**)			❶❷❸
theta	θ	Mobility modulation	0.0	V-*1*	❸
eta	η	Static feedback	0.0	—	❸
kappa	κ	Saturation field factor	0.2	—	❸
del		Channel length reduction per side	0.0	m	❶❷❸

Large-Signal Model



This schematic represents the Level 2 model. The Level 1 model has the same configuration, without the junction diodes.

The terminals (*G*, *D*, *S*, *B*) represent the gate, drain, source, and bulk connections, respectively, of the MOSFET. *Id* is the current flowing from the drain to the source as a function of (*Vgs*, *Vds*, *Vbs*). The ohmic resistance of the drain and source diffusion junctions are represented by *Rd* and *Rs*. A P-channel MOSFET can be modeled by reversing the polarity of (*Vgs*, *Vds*, *Vbs*), the current source *Id*, and the junction diodes.

When **nrs** and **nrd** are not given as options on the MOSFET device statement, T-Spice computes them by calculating the number of squares from the geometry of the junction and then multiplying them by **RSH** to obtain the drain and source resistors *Rd* and *Rs*.

Level 1 Equations

The Level 1 model is included for SPICE compatibility, but it is inaccurate when simulating circuits with analog characteristics. The original intent of the model was to provide an approximation for digital circuits that could be evaluated quickly, reducing the simulation time by as much as a factor of two. However, T-Spice generally precalculates current and charge tables and does not gain from this speedup. The Level 1 model should be avoided for most circuits.

Current

Current behavior is modeled by three equations, representing the three regions of MOSFET operation: *cutoff*, *linear*, and *saturation*.

Cutoff region: Vgs ð *Vth*

$$I_{ds} = 0 \tag{0.243}$$

Linear region: Vgs > *Vth* and (*Vgs* – *Vth*) > *Vds*

$$I_{ds} = \frac{\beta}{2} V_{ds}(2(V_{gs} - V_{th}) - V_{ds})(1 + \lambda V_{ds}) \tag{0.244}$$

Saturation region: Vgs > *Vth* and (*Vgs* – *Vth*) ≤ *Vds*

$$I_{ds} = \frac{\beta}{2}(V_{gs} - V_{th})^2(1 + \lambda V_{ds}) \tag{0.245}$$

In equations (0.244) and (0.245),

$$\beta = \frac{W - (2 \cdot WD)}{L - (2 \cdot LD)} \cdot KP \tag{0.246}$$

Charge

$$Q_g = C_{gb}V_{gb} + C_{gd}V_{gd} + C_{gs}V_{gs} \tag{0.247}$$

$$Q_d = C_{gd}V_{gd} + \int C_{db}(v)dv \tag{0.248}$$

$$Q_s = C_{gs}V_{gs} + \int C_{sb}(v)dv \tag{0.249}$$

$$Q_b = -(Q_g + Q_d + Q_s) \tag{0.250}$$

Capacitance

The Meyer gate capacitance model was replaced by a simplified model that provides conservation of charge.

$$C_{gb} = C_{gs} = C_{gd} = \frac{1}{3} WL \frac{\varepsilon_{ox}}{T_{ox}} \tag{0.251}$$

The junction capacitance for the drain and source is dependent on the drain-bulk and drain-source voltages.

$$C_{db} = C_{pn}(V_{bd}, AD, PD) \tag{0.252}$$

$$C_{sb} = C_{pn}(V_{bs}, AS, PS) \tag{0.253}$$

For reverse bias *Vpn* < *FC* · *PB*,

$$C_{pn}(V_{pn}, A, P) = \left(CJ \cdot \frac{A}{\left(1 - \frac{V_{pn}}{PB}\right)^{MJ}} \right) + \left(CJSW \cdot \frac{P}{\left(1 - \frac{V_{pn}}{PB}\right)^{MJSW}} \right) \qquad (0.254)$$

For forward bias *Vpn* ≥ *FC* · *PB*,

$$\begin{aligned} C_{pn}(V_{pn}, A, P) = {} & CJ\left(\frac{A}{(1 - FC)^{(1 + MJ)}}\right)\left(1 - FC1 + MJ + \frac{V_{pn}}{PB}MJ\right) + \\ & CJ\left(\frac{P}{(1 - FC)^{(1 + MJSW)}}\right)\left(1 - FC1 + MJSW + \frac{V_{pn}}{PB}MJSW\right) \end{aligned} \qquad (0.255)$$

In equations (0.254) and (0.255), *A* denotes area and *P* denotes perimeter.

The charge can be calculated using

$$Q = \int C(V) dv \qquad (0.256)$$

Threshold Voltage

An increase in the threshold voltage is due to a reverse bias from the gate to the substrate called the *body effect*, which causes degradation in the current drive of a transistor.

$$V_{to} = V_{FB} + (\gamma\sqrt{\phi} + \phi) \qquad (0.257)$$

$$\gamma = \frac{\sqrt{2\varepsilon_0\varepsilon_{si}qN_{sub}}}{C_{ox}} \qquad (0.258)$$

$$C_{ox} = \frac{\varepsilon_0\varepsilon_{ox}}{T_{ox}} \qquad (0.259)$$

$$KP = \mu_0 C_{ox} \qquad (0.260)$$

$$V_{FB} = \left(-TPG \cdot \frac{E_q}{2}\right) - \frac{\phi}{2} - \frac{qN_{ss}}{C_{ox}} \qquad (0.261)$$

Equation (0.257) is dependent on *Vbs* and is implemented as follows:

$$V_{th} = V_{to} + \gamma(bodyterm - \sqrt{2\phi_F}) \qquad (0.262)$$

When *Vbs* ≤ 0.0,

$$bodyterm = \sqrt{2\phi_F - V_{bs}} \qquad (0.263)$$

When *Vbs* > 0.0,

$$bodyterm = \max\left(\left(\sqrt{2\phi_F} - \frac{V_{bs}}{2\sqrt{2\phi_F}}\right),0\right) \tag{0.264}$$

In Equations (0.262) through (0.264),

$$2\phi_F = \phi = 2\frac{kT}{q}ln\left(\frac{N_{sub}}{n_i}\right) \tag{0.265}$$

Level 2 Equations

The Level 2 model uses two current equations. Since the MOSFET is not an ideal switch, the current begins to flow before the transistor reaches the turn-on voltage. This region is called *weak* inversion. As the voltage on the gate approaches the threshold voltage, it conducts current much more vigorously. At this point the channel is in the *strong* inversion region. The strong inversion region includes the linear and saturation regions. Many second-order effects that control the amount of current are calculated in the Level 2 model, including backgate bias and short and narrow channel effects. The Level 2 current equations are similar to the Grove equation for a MOSFET. These equations insure the continuity of current at *Von* (threshold point) through the transistor regions.

Subthreshold Region

The weak inversion current equation is used when the transistor is in the subthreshold region, *Vgs* < *Von*.

$$I_{ds} = \beta\left(\left(V_{on} - V_{bin} - \frac{\eta V_{on}}{2}\right)V_{on} - \frac{2}{3}\gamma_s((2\phi_F - V_{bs} + V_{on})^{3/2} - (2\phi_F - V_{bs})^{3/2})\right)e^{\frac{q}{nkT}(V_{gs} - V_{on})} \tag{0.266}$$

The strong inversion current equation is used to calculate the current when *Vgs* ≥ *Von*:

$$I_{ds} = \beta\left(\left(V_{gs} - V_{bin} - \frac{\eta V_{ds}}{2}\right)V_{ds} - \frac{2}{3}\gamma_s((2\phi_F - V_{bs} + V_{ds})^{3/2} - (2\phi_F - V_{bs})^{3/2})\right) \tag{0.267}$$

where *Vds* = *Vdsat* for *Vds* > *Vdsat*.

The voltages calculated to determined the second order effects of the transistor currents are defined as:

$$V_{th} = V_{bin} + \gamma_s\sqrt{2\phi_F - V_{bs}} \tag{0.268}$$

$$V_{bin} = V_{bi} + \delta\left(\frac{\pi\varepsilon_0\varepsilon_{si}}{4C_{ox}W_{eff}}\right)(2\phi_F - V_{bs}) \tag{0.269}$$

$$V_{bi} = V_{fb} + 2\phi_F \tag{0.270}$$

$$2\phi_F = \phi = 2\frac{kT}{q}\ln\left(\frac{N_{sub}}{n_i}\right) \tag{0.271}$$

$$V_{fb} = \phi_{ms} - \frac{q \cdot NSS}{C_{ox}} \tag{0.272}$$

The voltage *Von* is used to switch between the weak and strong inversion model regions. When *NFS* is not specified,

$$V_{on} = V_{th} \tag{0.273}$$

$$n = \infty \tag{0.274}$$

However, a more accurate cut-on point can be modeled by using a curve fitting parameter *NFS* in the evaluation of *Von*:

$$V_{on} = V_{th} + \frac{nkT}{q} \tag{0.275}$$

where

$$n = 1 + \frac{C_{FS}}{C_{ox}} + \frac{C_D}{C_{ox}} \tag{0.276}$$

$$C_{FS} = q \cdot NFS \tag{0.277}$$

$$C_D = \frac{\partial Q_B}{\partial V_{bs}} = C_{ox}\left(-\gamma_s\left(\frac{d\sqrt{2\phi_F - V_{bs}}}{dV_{bs}}\right)\right. \tag{0.278}$$
$$\left. - \left(\frac{\partial\gamma_s}{\partial V_{bs}}\right)\sqrt{2\phi_F - V_{bs}} + \delta\frac{\pi\varepsilon_0\varepsilon_{si}}{4C_{ox}W_{eff}}\right)$$

Taking the partial derivatives and applying the chain rule yields

$$C_D = C_{ox}\left[\frac{1}{2}\gamma_s\frac{1}{\sqrt{2\phi_F - V_{bs}}} - \gamma_s\sqrt{2\phi_F - V_{bs}}\left(\frac{X_D}{4L}\cdot\frac{1}{\left(1 + \frac{2X_D\sqrt{2\phi_F - V_{bs}}}{X_j}\right)^{1/2}}\cdot\frac{1}{\sqrt{2\phi_F - V_{bs}}}\right)\right] \qquad (0.279)$$

$$+\left(\frac{X_D}{4L}\cdot\frac{1}{\left(1 + \frac{2X_D\sqrt{2\phi_F - V_{bs} + V_{ds}}}{X_j}\right)^{1/2}}\cdot\frac{1}{\sqrt{2\phi_F - V_{bs} + V_{ds}}}\right) + \delta\frac{\pi\varepsilon_0\varepsilon_{si}}{4C_{ox}W_{eff}}$$

Equation (0.279) assumes that *Vbs* is negative. When *Vbs* is positive, $\sqrt{2\phi_F - V_{bs}}$ should be replaced with

$$\frac{\sqrt{2\phi_F}}{1 + \frac{V_{bs}}{4\phi_F}} \qquad (0.280)$$

Linear and Saturation Regions

The saturation voltage *Vdsat* can be computed with either of two methods. Which method is chosen depends on whether the input parameter *VMAX* has been defined.

Method 1. When *VMAX* is *not* defined, the model computes *Vdsat* assuming that the channel is pinched off at the drain:

$$V_{dsat} \qquad (0.281)$$

$$= \frac{V_{gs} - V_{bin}}{\eta} + \frac{1}{2}\left(\frac{\gamma_s}{\eta}\right)^2\left(1 - \sqrt{1 + 4\left(\frac{\eta}{\gamma_s}\right)^2\left(\frac{V_{gs} - V_{bin}}{\eta} + 2\phi_F - V_{bs}\right)}\right)$$

If the parameter λ is not defined while using the pinch-off method, then it can be computed as

$$\lambda = \frac{X_D}{L_l V_{ds}}\sqrt{\frac{V_{ds} - V_{dsat}}{4} + \sqrt{1 + \left(\frac{V_{ds} - V_{dsat}}{4}\right)^2}} \qquad (0.282)$$

The channel length becomes smaller due to the pinch-off of the channel. The effective channel length is

$$L_{eff} = L_l(1 - \lambda V_{ds}) \qquad (0.283)$$

Method 2. When *VMAX is* defined as *Vdsat* and the channel length modulation is calculated using the carrier scattering limited velocity model, this method appropriately models saturation current for short-channel MOSFETs as charge carriers reaching their maximum scattering limited velocity before the pinch-off effect comes into play.

$$VMAX = \frac{\mu_s\Bigl(\Bigl(V_{gs} - V_{bin} - \frac{\eta V_{dsat}}{2}\Bigr)V_{dsat} - \frac{2\gamma_s}{3}((V_{dsat} + 2\phi_F - V_{bs})^{3/2} - (2\phi_F - V_{bs})^{3/2})\Bigr)}{L_{eff}(V_{gs} - V_{bin} - \eta V_{dsat} - \gamma\sqrt{V_{dsat} + 2\phi_F - V_{bs}})} \qquad (0.284)$$

The effective channel length is dependent on *VMAX* and *Vdsat* and computed as

$$L_{eff} = L_1 - X_D\Bigl(\Bigl(\Bigl(\frac{X_D VMAX}{2\mu_s}\Bigr)^2 + (V_{ds} - V_{dsat})\Bigr)^{1/2} + \frac{(X_D)^2 VMAX}{2\mu_s} \qquad (0.285)$$

Solving for *Vdsat* is difficult because it requires the simultaneous solution of the nonlinear equations (0.284) and (0.285). A less computationally expensive approach is desirable:

Assume that *Leff* and *L1* in equation (0.285) are equal. Then by selecting parameter *NEFF*, adjust the value of *XD* to obtain a good fit to the I-V characteristics of the MOSFET.

$$X_D = \sqrt{\frac{2\varepsilon_0\varepsilon_{si}}{qN_{sub}NEFF}} \qquad (0.286)$$

Equation (0.285) can be solved using Ferrari's method. There will be 2 or 4 real roots. The smallest positive real root is the correct value of *Vdsat*.

If λ is not defined, then it can be computed using *Vdsat* and equations (0.284) and (0.287):

$$\lambda = \frac{X_D}{L_l V_{ds}}\Bigl(\Bigl(\Bigl(\Bigl(\frac{X_D VMAX}{2\mu_s}\Bigr)^2 + (V_{ds} - V_{dsat})\Bigr)^{1/2} + \frac{X_D VMAX}{2\mu_s}\Bigr) \qquad (0.287)$$

Then *Leff* can be computed as

$$L_{eff} = L_l(1 - \lambda V_{ds}) \qquad (0.288)$$

Thus the value of *Leff* is available from either of the two current saturation methods—channel pinch-off or carrier scattering limited velocity. If *Leff* is smaller than the zero-bias depletion layer width *WB* = *XD* $\sqrt{PB}$, then *Leff* must be recomputed as

$$L_{eff} = \frac{W_B}{1 + \frac{\Delta L - L_{max}}{W_B}} \qquad (0.289)$$

where

$$\Delta L = \lambda V_{ds} L_1 \qquad (0.290)$$

$$L_{max} = L_1 - W_B \qquad (0.291)$$

Equation (0.289) prevents numerical non-convergence, but does not model the punch-through effect.

The voltage *Vdsat* is compared with *Vds*. If *Vds* is greater than *Vdsat*, then *Vds* is set equal to *Vdsat*, thus clamping the drain-to-source voltage to *Vdsat* in the saturation region.

The result is a potential drop (*Vds*) across the saturated transistor and a smaller effective transistor gate length that increases the gain (β).

$$\beta = \frac{W_{eff}}{L_{eff}} KP \qquad (0.292)$$

$$KP = \mu_s C_{ox} \qquad (0.293)$$

KP is the transconductance of the MOSFET. If it is not specified, it will take the calculated value.

Second-Order Effects

Other variables, coefficients, and modified input parameters that are used in the model are defined as follows:

$$\phi_{ms} = -\left(\frac{2\phi_F}{2} + \frac{E_g}{2}\right) \tag{0.294}$$

$$C_{ox} = \frac{\varepsilon_0 \varepsilon_{ox}}{T_{ox}} \tag{0.295}$$

$$L_l = L - (2 \cdot LD) \tag{0.296}$$

$$W_{eff} = W - (2 \cdot WD) \tag{0.297}$$

$$\eta = 1 + \delta\left(\frac{\pi \varepsilon_0 \varepsilon_{si}}{4 C_{ox} W_{eff}}\right) \tag{0.298}$$

$$\mu_s = U_0\left(\frac{UCRIT \varepsilon_0 \varepsilon_{si}}{C_{ox}(V_{gs} - V_{on})}\right)^{UEXP} \tag{0.299}$$

$$\gamma_s = \gamma(1 - \alpha_s - \alpha_d) \tag{0.300}$$

$$\gamma = \frac{\sqrt{2 q \varepsilon_0 \varepsilon_{si} N_{sub}}}{C_{ox}} \tag{0.301}$$

$$\alpha_S = \frac{1}{2} \cdot \frac{X_j}{L_l} \cdot \left(\sqrt{1 + 2\frac{W_S}{X_J}} - 1\right) \tag{0.302}$$

$$\alpha_D = \frac{1}{2} \cdot \frac{X_j}{L_l} \cdot \left(\sqrt{1 + 2\frac{W_D}{X_J}} - 1\right) \tag{0.303}$$

$$W_S = X_D \sqrt{2\phi_F - V_{bs}} \tag{0.304}$$

$$W_D = X_D \sqrt{2\phi_F - V_{bs} + V_{ds}} \tag{0.305}$$

$$X_D = \sqrt{\frac{2\varepsilon_0 \varepsilon_{si}}{q N_{sub}}} \tag{0.306}$$

Ward-Dutton Charge Model

The Ward-Dutton charge model was used to replace the shortcomings of the Meyer capacitance model. It has been shown that assuming capacitance reciprocity leads to non-conservation of charge. This non-conservation of bogus charge can cause various charge-dependent circuits to be simulated incorrectly.

The Ward-Dutton model derives equations based on charge instead of capacitance and conserves charge through all regions of the transistor. This model applies to Level 2 and Level 3 MOSFET models only.

The following table lists the parameters used by the equations describing the Ward-Dutton charge model (see MOSFET Levels 4 and 13 (BSIM1) on page 395).

Symbol	***Description***	***Unit***
Idc	DC drain current	A
Vd	Drain voltage	V
Vg	Gate voltage	V
Vs	Source voltage	V
Vdsat	Saturation voltage	V
Qb	Total bulk charge	C
Qg	Total gate charge	C
Qs	Total source charge	C
Qd	Total drain charge	C
Qgi	Intrinsic gate charge	C
Qsi	Intrinsic source charge	C
Qdi	Intrinsic drain charge	C
Qgo	Overlap gate charge	C
Qso	Overlap source charge	C
Qdo	Overlap drain charge	C
Qsj	Junction source charge	C
Qdj	Junction drain charge	C

The Ward-Dutton model defines these normalized voltages as:

$$V_g = V_{gs} - V_{bs} - V_{fb} \tag{0.307}$$

$$V_d = V_{ds} - V_{bs} + 2\phi_F \tag{0.308}$$

$$V_s = -V_{bs} + 2\phi_F \tag{0.309}$$

The charge on the gate, drain, and source of a transistor is the total charge due to the overlap, junction, and intrinsic channel charge.

$$Q_g = Q_{gi} + Q_{go} \tag{0.310}$$

$$Q_d = Q_{di} + Q_{do} + Q_{dj} \tag{0.311}$$

$$Q_s = Q_{si} + Q_{so} + Q_{sj} \tag{0.312}$$

The overlap capacitors are due to the gate edges overlapping the substrate and diffusion junctions.

$$Q_{go} = C_{gso}W_{eff}V_{gs} + C_{gso}W_{eff}(V_{gs} - V_{ds}) + 2C_{gbo}L_l(V_{gs} - V_{bs}) \tag{0.313}$$

$$Q_{do} = C_{gdo}W_{eff}(V_{ds} - V_{gs}) \tag{0.314}$$

$$Q_{so} = -C_{gso}W_{eff}V_{gs} \tag{0.315}$$

The Ward-Dutton model addresses only the intrinsic channel charge. Two sets of equations model the regions of transistor charge.

Cut-off region. The charge model is in the cut-off region when the gate voltage is less than the threshold voltage.

When *Vgs* < *Vth* and (*Vgs* – *Vbs*) < *Vfb*,

$$Q_{di} = 0 \tag{0.316}$$

$$Q_{si} = 0 \tag{0.317}$$

$$Q_{gi} = C_{ox}W_{eff}L_lV_g \tag{0.318}$$

When *Vgs* < *Vth* and (*Vgs* – *Vbs*) > *Vfb*,

$$Q_{di} = 0 \tag{0.319}$$

$$Q_{si} = 0 \tag{0.320}$$

$$Q_{gi} = C_{ox}WL_l \cdot \frac{1}{2}\gamma_s\sqrt{\gamma_s^2 + 4V_g} \tag{0.321}$$

Linear and saturation regions. The model uses the same equation for the linear and saturation regions. In the saturation region the voltage *Vd* is clamped at the saturation voltage *Vdsat* when the following condition is true:

$$V_d + \gamma_s\sqrt{V_d} \ge V_g \tag{0.322}$$

Then

$$2\sqrt{V_{dsat}} = -\gamma_s + \sqrt{\gamma_s^2 + 4V_g} \tag{0.323}$$

$$V_d = (\sqrt{V_{dsat}})^2 \tag{0.324}$$

The intrinsic charge for the linear or saturation region is computed as

$$Q_{si} = \frac{3}{5}Q_{CSAT} + \frac{3}{10}(Q_C - Q_{CSAT}) \qquad (0.325)$$

$$Q_{di} = \frac{23}{5}Q_{CSAT} + \frac{7}{10}(Q_C - Q_{CSAT}) \qquad (0.326)$$

$$Q_C = -(Q_{gi} + Q_{bi}) \qquad (0.327)$$

$$Q_{CSAT} = -\frac{2}{3}C_{ox}W_{eff}L_1(V_g - V_s - \gamma_s\sqrt{V_s}) \qquad (0.328)$$

$$Q_{gi} = C_{ox}W_{eff}L_1\left[V_g - \frac{1}{I_{dc}}\left(V_g\frac{1}{2}(V_d^2 - V_s^2) - \frac{2}{5}\gamma_s(V_d^{5/2} - V_s^{5/2}) - \frac{1}{3}(V_d^3 - V_s^3)\right)\right] \qquad (0.329)$$

$$Q_{bi} = -\frac{C_{ox}W_{eff}L_1}{I_{dc}}\left[V_g\frac{2}{3}\gamma_s(V_d^{3/2} - V_s^{3/2}) - \gamma_s^2\frac{1}{2}(V_d^2 - V_s^2) - \frac{2}{5}\gamma_s(V_d^{5/2} - V_s^{5/2})\right] \qquad (0.330)$$

$$I_{dc} = V_g(V_d - V_s) - \frac{2}{3}\gamma_s(V_d^{3/2} - V_s^{3/2}) - \frac{1}{2}(V_d^2 - V_s^2) \qquad (0.331)$$

The diffusion junction capacitance equations for the drain and source are the same as in the Level 1 model. The capacitance is dependent on the drain to bulk or drain to source voltage.

For reverse bias $Vbs < FC \cdot PB$,

$$C_{bd} = CJ\left(\frac{AS}{\left(1 - \frac{V_{bs}}{PB}\right)^{MJ}}\right) + CJSW\left(\frac{PS}{\left(1 - \frac{V_{bs}}{PB}\right)^{MJSW}}\right) \qquad (0.332)$$

For forward bias $Vbs > FC \cdot PB$,

$$\begin{aligned} C_{bd} = {} & CJ\left(\frac{AS}{(1 - FC)^{(1 + MJSW)}}\right)\left[1 - FC(1 + MJ) + \frac{V_{bs}}{PB}MJ\right] + \\ & CJSW\left(\frac{PS}{(1 - FC)^{(1 + MJSW)}}\right)\left[1 - FC(1 + MJSW) + \frac{V_{bs}}{PB}MJSW\right] \end{aligned} \qquad (0.333)$$

To obtain the charge function at the junction of the drain or source, integrate the junction capacitance over the range of *Vbs*:

$$Q = \int C(v)dv \qquad (0.334)$$

Now integrate the reverse bias region from *Vbs* to ($FC \cdot PB$).

For reverse bias *Vbs* < *FC* · *PB*,

$$Q_{Dj} = \int_{V_{bs}}^{FC \cdot PB} C_{bd} V_{bs} dv \tag{0.335}$$

$$\begin{aligned} Q_{Dj} = {} & (CJ \cdot AS \cdot PB)\left(\left(\frac{1.0 - V_{bs}}{PB}\right)^{(1 - MJ)} - \frac{(1.0 - FC)^{(1 - MJ)}}{(1 - MJ)}\right) + \\ & (CJSW \cdot PS \cdot PB)\left(\left(\frac{1.0 - V_{bs}}{PB}\right)^{(1 - MJSW)} - \frac{(1.0 - FC)^{(1 - MJSW)}}{(1 - MJSW)}\right) \end{aligned} \tag{0.336}$$

For forward bias *Vbs* > *FC* · *PB*,

$$Q_{Dj} = \int_{FC \cdot PB}^{V_{bs}} C_{bd} V_{bs} dv \tag{0.337}$$

$$\begin{aligned} Q_{Dj} = {} & \frac{CJ \cdot AS \cdot PB}{(10 - FC)^{2 - MJ}} \cdot \Big[(V_{bs} - (FC \cdot PB))(1.0 - FC(2 - MJ)) + \\ & \frac{1}{2}\left(\frac{V_{bs}^2}{PB} - FC^2 PB\right)(1 - MJ)\Big] - \\ & \frac{CJ \cdot SW \cdot PS \cdot PB}{(1.0 - FC)^{2 + MJSW}} \cdot \Big[(V_{bs} - (FC \cdot PB))(1.0 - FC(2 - MJSW)) + \\ & \frac{1}{2}\left(\frac{V_{bs}^2}{PB} - FC^2 PB\right)(1 - MJ)\Big] \end{aligned} \tag{0.338}$$

Level 3 Equations

The Level 3 MOSFET model is a semi-empirical model developed to handle small-geometry devices.

The drain current in the linear and saturation regions (for *Vgs* > *Vth*) is calculated using

$$I'_{ds} = \beta \cdot \left[\frac{1}{1 + V_{ds}\left(\frac{\mu_s V_{max}}{L_1}\right)}\right] \cdot \left[V_{gs} - V_{th} - \left(\frac{1 + F_B}{2}\right) V_{ds}\right] \cdot V_{ds} \tag{0.339}$$

where *Vds* = *Vdsat* for *Vds* > *Vdsat* and *Vdsat* is defined as follows:

If *Vmax* ≤ 0.0,

$$V_{dsat} = \frac{V_{gs} - V_{th}}{1 + F_B} \tag{0.340}$$

If *Vmax* > 0.0,

$$V_{dsat} = \frac{V_{gs} - V_{th}}{1 + F_B} + \frac{V_{max} \cdot L_1}{\mu_s} - \sqrt{\left(\frac{V_{gs} - V_{th}}{1 + F_B}\right)^2 + \left(\frac{V_{max} \cdot L_1}{\mu_s}\right)^2} \tag{0.341}$$

and

$$\mu_s = \frac{\mu_0}{1 + \theta(V_{gs} - V_{th})} \tag{0.342}$$

$$\beta = \frac{W_{eff}}{L_1}\mu_s C_{ox} \tag{0.343}$$

$$F_B = \frac{\gamma F_s}{4\sqrt{2\phi_F - V_{bs}}} - F_n \tag{0.344}$$

$$F_s = 1 - \frac{X_j}{L_1}\left(\frac{LD + W_c}{X_j}\sqrt{1 - \left(\frac{W_p}{X_j + W_p}\right)^2} - \frac{LD}{X_j}\right) \tag{0.345}$$

$$W_p = X_d\sqrt{2\phi_F - V_{bs}} \tag{0.346}$$

$$X_d = \sqrt{\frac{2\varepsilon_{si}\varepsilon_0}{qN_A}} \tag{0.347}$$

$$\frac{W_c}{X_j} = 0.0631353 + 0.8013292\frac{W_p}{X_j} - 0.01110777\left(\frac{W_p}{X_j}\right)^2 \tag{0.348}$$

$$F_n = \frac{\varepsilon_{si}\varepsilon_0\delta\pi}{2C_{ox}W_{eff}} \tag{0.349}$$

$$V_{th} = V_{FB} + 2\phi_F - \sigma V_{ds} + \gamma F_s\sqrt{2\phi_F - V_{bs}} + F_n(2\phi_F - V_{bs}) \tag{0.350}$$

$$\sigma = \eta \cdot \frac{8.15 \times 10^{-22}}{C_{ox}L_1} \tag{0.351}$$

Channel length modulation is calculated for *Vds* > *Vdsat* as follows:

$$I_{dsat} = I'_{ds}\,(V_{ds} = V_{dsat}) \tag{0.352}$$

$$G_{dsat} = \frac{\partial I'_{ds}}{\partial V_{ds}}(V_{ds} = V_{dsat}) \tag{0.353}$$

$$I_{ds} = \frac{I_{dsat}}{1 - \left(\frac{L_1 - L'}{L_1}\right)} \tag{0.354}$$

where

$$L_1 - L' = \sqrt{\left(\frac{E_p X_D^2}{2}\right)^2 + K X_D^2 (V_{ds} - V_{dsat})} - \left(\frac{E_p X_D^2}{2}\right) \quad (0.355)$$

$$E_p = \frac{I_{dsat}}{G_{dsat} L_1} \quad (0.356)$$

For *Vgs* < *Von*, *Ids* is modified to included the weak inversion component of the drain current.

$$V_{on} = V_{th} + \frac{kT}{q} \cdot N \quad (0.357)$$

$$N = 1 + \frac{q}{C_{ox}} + \frac{\gamma F_s \sqrt{2\phi_F - V_{bs}} + F_n (2\phi_F - V_{bs})}{2(2\phi_F - V_{bs})} \quad (0.358)$$

$$I'_{ds} = I_{ds} \cdot e^{\frac{q}{NkT} \cdot (V_{gs} - V_{on})} \quad (0.359)$$

Temperature Dependence

The MOSFET model at Levels 1, 2, and 3 contains temperature-dependent model parameters. If a new temperature is specified with the **.temp** command, then these parameters must be modified before they are used in the current equations. The default temperature is 25 °C, which is equivalent to 300.15 K. The parameters affected by temperature changes are: ϕ*F* (Fermi potential), *Eg* (energy gap), *PB* (built-in potential of the drain and source), μ*0* (mobility), and *Is* (reverse current of the diffused junctions).

The energy gap between the conduction band and the valence band for polysilicon at *Tref* = 300.15 K and at the new temperature *Tnew* is:

$$E_{g,ref} = 1.16 - \left(\frac{7.02 \times 10^{-4} \cdot (T_{ref})^2}{1108 + T_{ref}}\right) \quad (0.360)$$

$$E_{g,new} = 1.16 - \left(\frac{7.02 \times 10^{-4} \cdot (T_{new})^2}{1108 + T_{new}}\right) \quad (0.361)$$

The intrinsic doping is adjusted by different equations depending whether the parameter *PHI* is defined. When *PHI* is *not* defined,

$$n_{i,new} = n_i \left(\frac{T_{new}}{T_{ref}}\right)^{3/2} \left(e^{\frac{q}{k}\left(\frac{E_{g,ref}}{T_{ref}} - \frac{E_{g,new}}{T_{new}}\right)}\right)^{1/2} \quad (0.362)$$

When *PHI is* defined,

$$n_{i,new} = N_{sub} \left(\frac{T_{new}}{T_{ref}}\right)^{3/2} \left(e^{\frac{q}{k}\left(\frac{E_{g,ref}}{T_{ref}} - \frac{E_{g,new}}{T_{new}} - \frac{2\phi_F}{T_{ref}}\right)}\right)^{1/2} \quad (0.363)$$

PHI for the new temperature is computed using *ni,new*:

$$\phi_{new} = 2\phi_{F,new} = \left(\frac{2kT_{new}}{q}\right)\ln\left(\frac{N_{sub}}{n_{i,new}}\right) \tag{0.364}$$

The conduction factor *KP* and the mobility vary with temperature as

$$\frac{\mu_{0,new}}{\mu_{0,ref}} = \left(\frac{T_{new}}{T_{ref}}\right)^{-3/2} \tag{0.365}$$

The parameter *KP* or β contains the temperature adjustment when computed from μ*0*; however, when *KP* is entered from the model statement, the value is modified as

$$\frac{\beta_{new}}{\beta_{ref}} = \left(\frac{T_{new}}{T_{ref}}\right)^{-3/2} \tag{0.366}$$

The MOSFET substrate junction diode saturation current varies as

$$\frac{I_{s,new}}{I_{s,ref}} = \left(\frac{T_{new}}{T_{ref}}\right)^{3}\left(e^{\frac{q}{k}\left(\frac{E_{g,ref}}{T_{ref}}-\frac{E_{g,new}}{T_{new}}\right)}\right) \tag{0.367}$$

The built-in potential *PB* is adjusted as

$$PB = PB\left(\frac{T_{new}}{T_{ref}}\right) - \left(\frac{2kT_{new}}{q}\right)\ln\left(\frac{T_{new}}{T_{ref}}\right)^{3/2} + \left[\left(\frac{T_{new}}{T_{ref}}\right)E_{g,ref} - E_{g,new}\right] \tag{0.368}$$

MOSFET Levels 4 and 13 (BSIM1)

Parameters

```
.model name nmos|pmos level= 4|13 [parameters]
```

Based on the Berkeley short-channel IGFET model, ©1990 Regents of the University of California. Also see Additional MOSFET Parameters on page 433.

Parameter	*Description*	*Default*	*Units*
vfb	Flat band voltage	–0.3	V
lvfb	Length sensitivity of **vfb**	0.0	µm · V
wvfb	Width sensitivity of **vfb**	0.0	µm · V
pvfb	*WL*-product sensitivity of **vfb**	0.0	$\mu m^2 \cdot V$
phi	Surface potential (double the Fermi potential)	0.7	V
lphi	Length sensitivity of **phi**	0.0	µm · V
wphi	Width sensitivity of **phi**	0.0	µm · V
pphi	*WL*-product sensitivity of **phi**	0.0	$\mu m^2 \cdot V$
k1	$\sqrt{V_{sb}}$ threshold coefficient	0.5	$V^{1/2}$
lk1	Length sensitivity of **k1**	0.0	$\mu m \cdot V^{1/2}$
wk1	Width sensitivity of **k1**	0.0	$\mu m \cdot V^{1/2}$
pk1	*WL*-product sensitivity of **k1**	0.0	$\mu m^2 \cdot V^{1/2}$
k2	Linear *Vsb* threshold coefficient	0.0	—
lk2	Length sensitivity of **k2**	0.0	µm
wk2	Width sensitivity of **k2**	0.0	µm
pk2	*WL*-product sensitivity of **k2**	0.0	μm^2
eta	Linear *Vds* threshold coefficient	0.0	—
leta	Length sensitivity of **eta**	0.0	µm
weta	Width sensitivity of **eta**	0.0	µm
peta	*WL*-product sensitivity of **eta**	0.0	μm^2
muz	Low drain field first-order mobility	600	$cm^2/(V \cdot s)$
lmuz	Length sensitivity of **muz**	0.0	$\mu m \cdot cm^2/(V \cdot s)$
wmuz	Width sensitivity of **muz**	0.0	$\mu m \cdot cm^2/(V \cdot s)$
pmuz	*WL*-product sensitivity of **muz**	0.0	$\mu m^2 \cdot cm^2/(V \cdot s)$

Parameter	*Description*	*Default*	*Units*
dl	Channel length reduction	0.0	µm
dw	Channel width reduction	0.0	µm
u0	Gate field mobility reduction	0.0	1/V
lu0	Length sensitivity of **u0**	0.0	µm/V
wu0	Width sensitivity of **u0**	0.0	µm/V
pu0	*WL*-product sensitivity of **u0**	0.0	µm^2/V
u1	Drain field mobility reduction	0.0	1/V
lu1	Length sensitivity of **u1**	0.0	µm/V
wu1	Width sensitivity of **u1**	0.0	µm/V
pu1	*WL*-product sensitivity of **u1**	0.0	µm^2/V
x2mz	*Vsb* correction to low-field 1st-order mobility	0.0	cm^2/(V^2·s)
lx2mz	Length sensitivity of **x2mz**	0.0	µm · cm^2 /(V^2 · s)
wx2mz	Width sensitivity of **x2mz**	0.0	µm · cm^2 /(V^2 · s)
px2mz	*WL*-product sensitivity of **x2mz**	0.0	µm^2·cm^2 /(V^2 · s)
x2e	*Vsb* correction to linear *Vds* threshold coefficient	0.0	1/V
lx2e	Length sensitivity of **x2e**	0.0	µm/V
wx2e	Width sensitivity of **x2e**	0.0	µm/V
px2e	*WL*-product sensitivity of **x2e**	0.0	µm^2/V
x3e	*Vds* correction to linear *Vds* threshold coefficient	0.0	1/V
lx3e	Length sensitivity of **x3e**	0.0	µm/V
wx3e	Width sensitivity of **x3e**	0.0	µm/V
px3e	*WL*-product sensitivity of **x3e**	0.0	µm^2/V
x2u0	*Vsb* reduction to gate field mobility reduction	0.0	1/V^2
lx2u0	Length sensitivity of **x2u0**	0.0	µm/V^2
wx2u0	Width sensitivity of **x2u0**	0.0	µm/V^2
px2u0	*WL*-product sensitivity of **x2u0**	0.0	µm^2/V^2
x2u1	*Vsb* reduction to drain field mobility reduction	0.0	1/V^2
lx2u1	Length sensitivity of **x2u1**	0.0	µm/V^2
wx2u1	Width sensitivity of **x2u1**	0.0	µm/V^2
px2u1	*WL*-product sensitivity of **x2u1**	0.0	µm^2/V^2

Parameter	*Description*	*Default*	*Units*
mus	High drain field mobility	600	$\mathrm{cm}^2/(\mathrm{V}\cdot\mathrm{s})$
lmus	Length sensitivity of **mus**	0.0	$\mu\mathrm{m}\cdot\mathrm{cm}^2/(\mathrm{V}\cdot\mathrm{s})$
wmus	Width sensitivity of **mus**	0.0	$\mu\mathrm{m}\cdot\mathrm{cm}^2/(\mathrm{V}\cdot\mathrm{s})$
pmus	*WL*-product sensitivity of **mus**	0.0	$\mu\mathrm{m}\cdot\mathrm{cm}^2/(\mathrm{V}\cdot\mathrm{s})$
x2ms	*Vbs* reduction to high drain field mobility	0.0	$\mathrm{cm}^2/(\mathrm{V}^2\cdot\mathrm{s})$
lx2ms	Length sensitivity of **x2ms**	0.0	$\mu\mathrm{m}\cdot\mathrm{cm}^2/(\mathrm{V}^2\cdot\mathrm{s})$
wx2ms	Width sensitivity of **x2ms**	0.0	$\mu\mathrm{m}\cdot\mathrm{cm}^2/(\mathrm{V}^2\cdot\mathrm{s})$
px2ms	*WL*-product sensitivity of **x2ms**	0.0	$\mu\mathrm{m}\cdot\mathrm{cm}^2/(\mathrm{V}\cdot\mathrm{s})$
x3ms	*Vds* reduction to high drain field mobility	5.0	$\mathrm{cm}^2/(\mathrm{V}^2\cdot\mathrm{s})$
lx3ms	Length sensitivity of **x3ms**	0.0	$\mu\mathrm{m}\cdot\mathrm{cm}^2/(\mathrm{V}^2\cdot\mathrm{s})$
wx3ms	Width sensitivity of **x3ms**	0.0	$\mu\mathrm{m}\cdot\mathrm{cm}^2/(\mathrm{V}^2\cdot\mathrm{s})$
px3ms	*WL*-product sensitivity of **x3ms**	0.0	$\mu\mathrm{m}^2\cdot\mathrm{cm}^2/(\mathrm{V}^2\cdot\mathrm{s})$
x3u1	*Vds* reduction to drain field mobility reduction	0.0	$1/\mathrm{V}^2$
lx3u1	Length sensitivity of **x3u1**	0.0	$\mu\mathrm{m}/\mathrm{V}^2$
wx3u1	Width sensitivity of **x3u1**	0.0	$\mu\mathrm{m}/\mathrm{V}^2$
px3u1	*WL*-product sensitivity of **x3u1**	0.0	$\mu\mathrm{m}^2/\mathrm{V}^2$
tox	Gate oxide thickness	0.02	μm; Å if >1
temp	Temperature	25.0	°C
vdd	Critical voltage for high drain field mobility reduction	5.0	V
cgdo	Gate/drain parasitic capacitance per unit channel width	1.5×10^{-9}	F/m
cgso	Gate/source parasitic capacitance per unit channel width	1.5×10^{-9}	F/m
cgbo	Gate/bulk parasitic capacitance per unit channel length	2.0×10^{-10}	F/m
xpart	Flag for channel charge partitioning	1	—

Parameter	*Description*	*Default*	*Units*
n0	Low-field weak inversion gate drive coefficient. A value □≥ 200 disables weak inversion calculation.	0.5	—
ln0	Length sensitivity of **n0**	0.0	µm
wn0	Width sensitivity of **n0**	0.0	µm
pn0	*WL*-product sensitivity of **n0**	0.0	µm^2
nb	*Vsb* reduction to **n0**	0.0	—
lnb	Length sensitivity of **nb**	0.0	µm
wnb	Width sensitivity of **nb**	0.0	µm
pnb	*WL*-product sensitivity of **nb**	0.0	µm^2
nd	*Vds* reduction to **n0**	0.0	—
lnd	Length sensitivity of **nd**	0.0	µm
wnd	Width sensitivity of **nd**	0.0	µm
pnd	*WL*-product sensitivity of **nd**	0.0	µm^2
xl | dl | ldel	Mask and etching length change	0.0	m
xw | dw | wdel	Mask and etching width change	0.0	m

MOSFET Level 28 (Extended BSIM1)

```
.model name nmos|pmos level=28 [parameters]
```

T-Spice supports MOSFET model level 28, based on the Berkeley short-channel IGFET models. The Extended BSIM1 level 28 model is a proprietary Tanner Research extension to the core BSIM1 model developed at Berkeley. The extensions attempt to fix many of the problems in the original model equations, including:

- Negative output conductance in the saturation region of operation.
- Discontinuities in the output conductance at the transition between the linear and saturation regions of operation.
- Discontinuities in the subthreshold current and transconductance characteristics near threshold.

In addition, the core equations have been enhanced to correct deficiencies in the original model equations:

- Temperature compensation equations have been added for proper handling of temperature effects.
- Effective length-width product scaling factors have been added for use with the BSIM3 revision 3 scaling equations.

Note: T-Spice accepts BSIM1 model parameters entered in the level 13 and level 28 conventions used by HSPICE™*.
Level 13 model parameters are translated to standard BSIM1 model (see MOSFET Levels 4 and 13 (BSIM1) on page 395.)
Level 28 model parameters are automatically translated to the extended BSIM1 model as shown below.

Parameters

Following are the Tanner Extended BSIM1 MOSFET model parameters.

Parameter	*Description*	*Default*	*Units*
tempmod	Temperature model selector. **tempmod=0** disables the temperature compensation equation. **tempmod>0** enables the temperature compensation equation.	0	—
ute	Temperature exponent for mobility	1.5	—
lute	Length sensitivity of **ute**	0.0	—
wute	Width sensitivity of **ute**	0.0	—
pute	*WL*-product sensitivity of **ute**	0.0	—
kt1	Temperature coefficient for flat band voltage	0.0	V
lkt1	Length sensitivity of **kti**	0.0	μm·V
wkt1	Width sensitivity of **kti**	0.0	μm·V
pkt1	*WL*-product sensitivity of **kti**	0.0	$\mu m^2 \cdot V$

* HSPICE(®) is a registered trademark of Synopsys, Inc.

Parameter	*Description*	*Default*	*Units*
submod	Subthreshold current model selector	0	—
voffset	Subthreshold offset voltage above threshold	0.0	V
lvoffset	Length sensitivity of **voffset**	0.0	μm·V
wvoffset	Width sensitivity of **voffset**	0.0	μm·V
pvoffset	*WL*-product sensitivity of **voffset**	0.0	$\mu m^2 \cdot V$
iratio	Subthreshold current factor	e^{-2}	—
liratio	Length sensitivity of **iratio**	0.0	—
wiratio	Width sensitivity of **iratio**	0.0	—
piratio	*WL*-product sensitivity of **iratio**	0.0	—
satmod	Saturated drain current model selector	0	—
alpha	Saturated drain current knee parameter	1.0	—
lalpha	Length sensitivity of **alpha**	0.0	—
walpha	Width sensitivity of **alpha**	0.0	—
palpha	*WL*-product sensitivity of **alpha**	0.0	—
gamma	Saturated drain current output conductance parameter	0.0	—
lgamma	Length sensitivity of **gamma**	0.0	—
wgamma	Width sensitivity of **gamma**	0.0	—
pgamma	*WL*-product sensitivity of **gamma**	0.0	—
xj	Junction depth	0.0	m
wmlt	Width shrink factor	1.0	—

Equations

Process Parameters

Process parameters express the sensitivity of the BSIM1 electrical parameters to device length, width, and the product of length and width (*WL*-product). The SPICE names of process parameters are derived from the related electrical parameter names by prefixing the letters **l**, **w**, and **p**.

The actual value of an electrical parameter *P* is

$$P = P_0 + \frac{P_L}{L_i - \Delta L} + \frac{P_W}{W_i - \Delta W} + \frac{P_P}{(L_i - \Delta L)(W_i - \Delta W)} \tag{0.369}$$

where *PL*, *PW*, and *PP* denote *P*’s length, width, and product sensitivity parameters, respectively.

Drain Current

Drain current is modeled by three equations, representing the three regions of MOSFET operation: *cutoff*, *linear*, and *saturation*.

Cutoff region: Vgs ≤ Vth

$$I_{ds} = 0 \tag{0.370}$$

Linear region: Vgs > Vth and 0 < *Vds* < *Vd,sat*

$$I_{ds} = \frac{\mu_0}{1 + U_0(V_{gs} - V_{th})} \cdot \frac{C'_{ox}\dfrac{W_{eff}}{L_{eff}}}{\left(1 + \dfrac{U_1}{L_{eff}}V_{ds}\right)} \cdot \left[(V_{gs} - V_{th})V_{ds} - \frac{a}{2}V_{ds}^2\right] \tag{0.371}$$

where

$$a = \frac{1 + gK_1}{2\sqrt{\phi_s - V_{bs}}} > 1 \tag{0.372}$$

and

$$g = 1 - \frac{1}{1.744 + 0.8364(\phi_s - V_{bs})} \tag{0.373}$$

Saturation region: Vgs > Vth and *Vds ≥ Vd,sat*

$$I_{ds} = \frac{\mu_0}{1 + U_0(V_{gs} - V_{th})} \cdot \frac{C'_{ox}\dfrac{W_{eff}}{L_{eff}}}{2aK} \cdot (V_{gs} - V_{th})^2 \tag{0.374}$$

where

$$K = \frac{1 + v_c + \sqrt{1 + 2v_c}}{2} \tag{0.375}$$

and

$$v_c = \frac{U_1}{L_{eff}} \cdot \frac{(V_{gs} - V_{th})}{a} \tag{0.376}$$

In the linear and saturation regions,

$$V_{d,sat} = \frac{V_{gs} - V_{th}}{a\sqrt{K}} \tag{0.377}$$

$$U_0(V_{ds}, V_{bs}) = U_{0z} + U_{0b}V_{bs} \tag{0.378}$$

$$U_1(V_{ds}, V_{bs}) = U_{1z} + U_{1b}V_{bs} + U_{1d}(V_{ds} - V_{dd}) \tag{0.379}$$

μ*0* is computed by quadratic interpolation given three conditions:

$$\mu_0(V_{ds} = 0) = \mu_z + \mu_{zb} V_{bs} \tag{0.380}$$

$$\mu_0(V_{ds} = V_{dd}) = \mu_s + \mu_{sb} V_{bs} \tag{0.381}$$

and the sensitivity of μ*0* to the drain bias at *Vds* = *Vdd*.

Subthreshold Current

The total drain current is modeled as the sum of two components: the *strong inversion* component *Ids,s*, equivalent to the drain current modeled in equations (0.370) through (0.381), and the weak inversion component *Ids,w*:

$$I_{ds,w} = \frac{I_{exp} I_{limit}}{I_{exp} + I_{limit}} \tag{0.382}$$

where

$$I_{exp} = \mu_0 C'_{ox} \frac{W_{eff}}{L_{eff}} \left(\frac{kT}{q}\right)^2 e^{1.8} e^{q(V_{gs} - V_{th})/nkT} (1 - e^{-qV_{ds}/kT}) \tag{0.383}$$

and

$$I_{limit} = \frac{\mu_0 C'_{ox}}{2} \frac{W_{eff}}{L_{eff}} \left(\frac{3kT}{q}\right)^2 \tag{0.384}$$

The subthreshold parameter *n* is modeled as

$$n(V_{ds}, V_{bs}) = n_0 + n_b V_{bs} + n_d V_{ds} > 0.5 \tag{0.385}$$

MOSFET Level 5 (Maher-Mead)

The Maher-Mead MOSFET model is accurate, physically based, continuous over all transistor regions of operation, including subthreshold, and scales to submicron channel lengths.

Parameters

```
.model name nmos|pmos level=5 [parameters]
```

Also see Additional MOSFET Parameters on page 433.

Parameter	*Symbol*	*Description*	*Default*	*Unit*
solver	*solver*	Nonlinear system solver selector: bisection: **solver**=0 secant: **solver**=1	1	—
tox	*Tox*	Oxide thickness	1.0e-7	m
vmax	*Vmax*	Saturated velocity of electrons or holes	0	m ⁄s
mu0	*µ0*	Zero gate field mobility	Computed.	$cm^2⁄V\cdot s$
nsub	*Nsub*	Substrate doping	—	cm^{-3}
vfb	*Vfb*	Flat band voltage	0.0	V
eghalf	*Eghalf*	Electric field where mobility = **mu0**	1.0×10^{10}	V ⁄m
ld	*Ld*	Length adjustment parameter	0.75 × *xj*	m
wd	*Wd*	Width adjustment parameter	0.0	m
xl | dl | ldel	*Xl*	Mask and etching length change	0.0	m
xw | dw | wdel	*Xw*	Mask and etching width change	0.0	m
xj	*Xj*	Junction depth	0.0	m
del	*del*	Channel length reduction per side	0.0	m
tnom | tref	*Tref*	Reference temperature	global **tnom** (25.0)	°C
qstol	*qstol*	Tolerance for nonlinear solution of source charge	1.0×10^{-8}	
qdtol	*qdtol*	Tolerance for nonlinear solution of drain charge	1.0×10^{-8}	
qdmindydx		Lower bound on derivative in nonlinear solution of drain charge	0.01	

Characteristics

The Maher-Mead model is a physically based, charge-controlled model for the DC current, the intrinsic terminal charges, and the transcapacitances in the MOSFET.

The model expresses the current in the MOSFET in terms of the mobile charge per unit area in the channel, and uses a complete set of natural units for velocity, voltage, length, charge, and current. The current-flow equation for the transistor includes both a drift term and a diffusion term, so that the formulation applies equally over the subthreshold, saturation, and “ohmic” regions of transistor operation and includes the effect of velocity saturation.

The model uses physical parameters derived from the fabrication process by direct measurement and from the dimensions of the device. The model agrees closely with measurements on the scaling of current with channel length down to submicron channel lengths.

MOSFET Level 47 (BSIM3 Revision 2)

Parameters

```
.model name nmos|pmos level=47 [parameters]
```

Based on the Berkeley short-channel IGFET model, ©1990 Regents of the University of California.]

Also see Additional MOSFET Parameters on page 433.

Parameter	*Symbol*	*Description*	*Default*	*Units*
subthmod	*subthmod*	Subthreshold model selector	2	—
satmod	*satmod*	Saturation model selector	2	—
bulkmod	*bulkmod*	Bulk charge effect model selector	1 [n] 2 [p]	—
mobmod	*mobmod*	Mobility model selector	1	—
tox	*Tox*	Gate oxide thickness	1.50×10^{-8}	m
cdsc	*Cdsc*	Drain/source and channel coupling capacitance	2.4×10^{-4}	F/m^2
cdscb	*Cdscb*	Body effect coefficient of **cdsc**	0.0	$F/V \cdot m^2$
cit	*Cit*	Interface state capacitance	0.0	F/m^2
nfactor	*Nfactor*	Swing coefficient	1	—
xj	*Xj*	Junction depth	1.50e-7	m
vsat	*νsat*	Saturation velocity at **tnom**	8.0×10^{-6}	cm/s
at	*At*	Temperature coefficient of **vsat**	3.3×10^{-4}	m/s
a0	*A0*	Non-uniform depletion width effect coefficient	1.0 [**bulkmod**=1] 4.4 [**bulkmod**=2]	—
a1	*A1*	Non-saturation factor 1	0.0 [n] 0.23 [p]	1/V
a2	*A2*	Non-saturation factor 2	1.0 [n] 0.08 [p]	—
keta	*Keta*	Body bias coefficient of non-uniform depletion width effect	–0.047	1/V
vghigh	*Vghigh*	High bound of transition region	0.12	V
vglow	*Vglow*	Low bound of transition region	–0.12	V
nsub	*Nsub*	Doping concentration	6.0×10^{16}	cm^{-3} (≤10^{20}) m^{-3} (>10^{20})

Parameter	*Symbol*	*Description*	*Default*	*Units*
nch | npeak	*Npeak*	Peak doping concentration	1.7×10^{17}	cm^{-3} ($\le 10^{23}$) m^{-3} ($>10^{23}$)
ngate	*Ngate*	Gate doping concentration	0.0	cm^{-3}
gamma1	$\gamma 1$	*Vth* coefficient	0.0	$V^{1/2}$
gamma2	$\gamma 2$	*Vth* coefficient 2	0.0	$V^{1/2}$
vbx	*Vbx*	*Vth* transition body voltage	0.0	V
vbi	*Vbi*	Drain/source junction built-in potential	0.0	V
vbm	*Vbm*	Maximum body voltage	–5.0	V
xt	*Xt*	Doping depth	1.55×10^{-7}	m
phi	ϕ	Strong inversion surface potential	Computed	V
litl	*Litl*	Depth of current path	0.0	m
em	*Em*	Maximum electric field	4.1×10^{7}	V/m
k1	*K1*	Bulk effect coefficient 1	0.0	$V^{1/2}$
kt1	*Kt1*	Temperature coefficient of *Vth*	–0.11	V
kt1l	*Kt1l*	Channel length sensitivity of **kt1**	0.0	V·m
kt2	*Kt2*	Body bias coefficient of **kt1**	0.022	—
k2	*K2*	Bulk effect coefficient 2	0.0	—
k3	*K3*	Narrow width effect coefficient	80.0	—
k3b	*K3b*	Body effect coefficient of **k3**	0.0	1/V
w0	*W0*	Narrow width effect coefficient	2.5×10^{-6}	m
nlx	*Nlx*	Lateral non-uniform doping effect	1.74×10^{-7}	m
dvt0	*Dvt0*	Short channel effect coefficient 0	2.2	—
dvt1	*Dvt1*	Short channel effect coefficient 1	0.53	—
dvt2	*Dvt2*	Short channel effect coefficient 2	–0.032	1/V
drout	*DRout*	DIBL effect on *Rout* coefficient	0.56	—
dsub	*Dsub*	DIBL effect coefficient in subthreshold region	**drout**	—
vtho | vth0	*Vth*	Threshold voltage	0.7 [n] –0.7 [p]	V
ua	*Ua*	Linear *Vgs* dependence of mobility	2.25×10^{-9}	m/V
ua1	*Ua1*	Temperature coefficient of **ua**	4.31×10^{-9}	m/V
ub	*Ub*	Quadratic *Vgs* dependence of mobility	5.87×10^{-19}	m^2/V^2
ub1	*Ub1*	Temperature coefficient of **ub**	-7.61×10^{-18}	m^2/V^2
uc	*Uc*	Body-bias dependence of mobility	0.0465	1/V
uc0	*Uc0*	Mobility coefficient	0.0	V^2/m^2

Parameter	*Symbol*	*Description*	*Default*	*Units*
uc1	*Uc1*	Temperature coefficient of **uc**	–0.056	1/V
u0	µ*0*	Low-field mobility at **tnom**	670.0 [n] 250.0 [p]	cm^2/V·s (≥ 1) m^2/V·s (< 1)
ute	µ*te*	Temperature coefficient of mobility	–1.5	—
voff	*Voff*	Threshold voltage offset	–0.11	V
vfb	*Vfb*	Flat band voltage	–1.0	V
dl | **ld**	*Dl*	Channel length reduction	0.0	m
dw | **wd**	*Dw*	Channel width reduction	0.0	m
lmlt	*Lmlt*	Length shrink factor	1.0	—
wmlt	*Wmlt*	Width shrink factor	1.0	—
xl | **dl** | **ldel**	*Xl*	Mask and etching length reduction factor	0.0	—
xw | **dw** | **wdel**	*Xw*	Mask and etching width reduction factor	0.0	—
tnom | **tref**	*Tnom*	Temperature	25.0	°C
cgso | **cgsom**	*Cgso*	Gate/source overlap capacitance per unit channel width	0.0	F/m
cgdo | **cgdom**	*Cgdo*	Gate/drain overlap capacitance per unit channel width	0.0	F/m
cgbo | **cgbom**	*Cgbo*	Gate/bulk overlap capacitance per unit channel length	0.0	F/m
xpart	*Xpart*	Flag for channel charge partitioning	0.0	—
rdsw	*Rdsw*	Source/drain resistance per unit width	0.0	Ω·µm
rds0	*Rds0*	Source/drain contact resistance	0.0	Ω
ldd	*LDD*	Total source/drain LDD region length	0.0	m
eta	*Eta*	Effective drain voltage coefficient	0.3	—
eta0	*Eta0*	Subthreshold region DIBL coefficient	0.08	—
etab	*Etab*	Subthreshold region DIBL coefficient	–0.07	1/V
pclm	*Pclm*	Channel-length modulation effect coefficient	1.3	—
pdibl1	*Pdibl1*	DIBL effect coefficient 1	0.39	—
pdibl2	*Pdibl2*	DIBL effect coefficient 2	0.0086	—
pscbe1	*Pscbe1*	Substrate current body effect coefficient 1	4.24×10^{8}	V/m
pscbe2	*Pscbe2*	Substrate current body effect coefficient 2	1.0×10^{-5}	m/V
pvag	*Pvag*	*Vg* dependence of *Rout* coefficient	0.0	—

Equations

For the complete equations describing the Level 47 model, refer to J. H. Huang, Z. H. Liu, M. C. Jeng, K. Hui, M. Chan, P. K. Ko, and C. Hu, BSIM3 Manual (Version 2.0). Berkeley, CA: University of California, 1994.).

Drain Current

In the linear region:

$$I_{ds} = \frac{I_{dslin0}}{1 + \frac{R_{ds} I_{dslin0}}{V_{ds}}} \tag{0.386}$$

where

$$I_{dslin0} = \mu_{eff} C_{ox} \left(\frac{W}{L}\right) \left(\frac{1}{1 + \frac{V_{ds}}{E_{sat} L}}\right) \left(V_{gst} - A_{bulk} \frac{V_{ds}}{2}\right) V_{ds} \tag{0.387}$$

In the saturation region:

$$I_{ds} = I_{dsat} \left(1 + \frac{V_{ds} - V_{dsat}}{V_A}\right) \left(1 + \frac{V_{ds} - V_{dsat}}{V_{ASCBE}}\right) \tag{0.388}$$

where

$$I_{dsat} = W \nu_{sat} C_{ox} (V_{gst} - A_{bulk} V_{dsat}) \tag{0.389}$$

In the subthreshold region:

If **subthmod** = 0:

$$I_{ds} = 0 \tag{0.390}$$

If **subthmod** = 1:

$$I_{ds} = \frac{I_{limit} I_{exp}}{I_{limit} + I_{exp}} (1 - e^{-V_{ds} / V_{tm}}) \tag{0.391}$$

where

$$I_{limit} = \frac{9}{2}\mu_0 C_{dep}\left(\frac{W}{L}\right)V_{tm}^2 \cdot e^{\left(\frac{(E_{ta0} + E_{tab} \cdot V_{bs})\theta_{dibl}V_{ds}}{nV_{tm}}\right)} \tag{0.392}$$

$$I_{exp} = \mu_0 C_{dep}\left(\frac{W}{L}\right)V_{tm}^2 \cdot e^{\left(\frac{V_{gs} - V_{th} - V_{off} + (E_{ta0} + E_{tab} \cdot V_{bs})\theta_{dibl}V_{ds}}{nV_{tm}}\right)} \tag{0.393}$$

If **subthmod** = 2:

$$I_{ds} = I_{s0}(1 - e^{-V_{ds}/V_{tm}}) \cdot e^{\left(\frac{V_{gst} - V_{off} + (E_{ta0} + E_{tab} \cdot V_{bs})\theta_{dibl}V_{ds}}{nV_{tm}}\right)} \tag{0.394}$$

where

$$I_{s0} = \mu_0 \frac{W}{L}\sqrt{\frac{q\varepsilon_{si}N_{peak}}{2\phi_s}} \cdot V_{tm}^2 \tag{0.395}$$

In the transition region:

$$I_{ds} = (1 - t)^2 I_{dslow} + 2(1 - t)tI_p + t^2 I_{dshigh} \tag{0.396}$$

where

$$t = \left(\frac{V_p - V_{gslow}}{V_{gslow} - 2V_p + V_{gshigh}}\right) \times \left(\sqrt{1 + \frac{(V_{gslow} - 2V_p + V_{gshigh})(V_{gst} - V_{gslow})}{(V_p - V_{gslow})^2}} - 1\right) \tag{0.397}$$

Variables for which equations are not given here are as follows.

VA	Early voltage
VASCBE	Early voltage due to substrate current-induced body effect
μ*eff*	Effective mobility
Abulk	Bulk charge effect factor
Vdsat	Drain saturation voltage

MOSFET Levels 8, 49 and 53 (BSIM3 Revision 3.3)

Parameters

```
.model name nmos|pmos level=8|49|53 [parameters]
```

Levels 49 and 53 are based upon the 3.3 version of Berkeley SPICE. They contain the most commonly used HSPICE ACM, parasitic resistor, and parasitic diode extensions. These extensions are described in Additional MOSFET Parameters on page 433.

Level 8 is a strict Berkeley v3.30 implementation with the addition of the HSPICE Effective area and Perimeter calculations and parasitic resistor equations, but not the diode equations.

Refer to the Berkeley manual **BSIM3v330.pdf** for further model parameters and equations.

Model Selectors

Parameter	*Symbol*	*Description*	*Default*
binflag	binflag	Flag for use of **xwref** and **xlref** parameters	0 (Off)
capmod	*capmod*	Flag for short channel capacitance model	1 [v3.0] 0 [v3.1] 3 [≥v3.2]
mobmod	*mobmod*	Mobility model selector.	1
nqsmod	*nqsmod*	Flag for NQS model. This turns the non-quasistatic model equations on or off, and overrides the model parameter value. ***Note:*** This option applies to BSIM3 v3.2 and v3.3 only. **nqsmod** is a device parameter as well as a device statement.	0
version	version	Select version of Berkeley BSIM3: 3.0, 3.1. 3.2, or 3.3.	3.3

Basic Model Parameters

Parameter	*Symbol*	*Description*	*Default*	*Units*
a1	*A1*	Non-saturation factor 1	0.0 [n] 0.23 [1] [p]	V^{-1}
a2	*A2*	Non-saturation factor 2	1.0 [n] 0.08 [p]	
ags	*Ags*	Gate bias coefficient of *Abulk*	0.0	V^{-1}
alpha0	$\alpha 0$	1st parameter of impact ionization current	0	m/V

Parameter	***Symbol***	***Description***	***Default***	***Units***
b0	*B0*	Bulk charge effect coefficient for channel width	0.0	m
b1	*B1*	Bulk charge effect width offset	0.0	m
beta0	β*0*	2nd parameter of impact ionization current	30	V
cdsc	*Cdsc*	Drain/source and channel coupling capacitance	2.4×10^{-4}	F/m^2
cdscb	*Cdscb*	Body effect coefficient of **cdsc**	0.0	$F/V \cdot m^2$
cdscd	*Cdscd*	Drain-bias sensitivity of **cdsc**	0.0	$F/V \cdot m^2$
cit	*Cit*	Interface state capacitance	0.0	F/m^2
delta	δ	Effective *Vds* parameter	0.01	—
drout	*DRout*	DIBL effect on *Rout* coefficient	0.56	—
dsub	*Dsub*	DIBL effect coefficient in subthreshold region	**drout**	
dvt0	*Dvt0*	Short channel effect coefficient 0	2.2	
dvt0w	*Dvt0w*	1st coefficient of narrow width effect on **vth** at small *L*	0	
dvt1	*Dvt1*	Short channel effect coefficient 1	0.53	
dvt1w	*Dvt1w*	2nd coefficient of narrow width effect on **vth** at small *L*	5.3×10^{6}	m^{-1}
dvt2	*Dvt2*	Short channel effect coefficient 2	–0.032	V^{-1}
dvt2w	*Dvt2w*	Body-bias coefficient of narrow width effect on **vth** at small *L*	–0.032	V^{-1}
eta0	η*0*	Subthreshold region DIBL coefficient	0.08	
etab	η*b*	Subthreshold region DIBL coefficient	–0.07 V^{-1}	V^{-1}
is	I_s	Bulk saturation current	1.0×10^{-14}	A
k1	*K1*	1st order bulk effect coefficient	0.53	$V^{1/2}$
k2	*K2*	2nd order bulk effect coefficient	-0.0186	
k3	*K3*	Narrow width effect coefficient	80.0	
k3b	*K3b*	Body effect coefficient of **k3**	0.0	V^{-1}
keta	K_η	Body bias coefficient of non-uniform depletion width effect	–0.047	V^{-1}

Parameter	*Symbol*	*Description*	*Default*	*Units*
nch | npeak	*Nch*	Peak doping concentration	1.7×10^{17}	cm^{-3} Note: T-Spice assigns units of m^{-3} to values > 10^{23}.
nfactor	*Nfactor*	Subthreshold swing coefficient	1	—
ngate	*Ngate*	Poly gate doping concentration	0	cm^{-3}
nlx	*Nlx*	Lateral non-uniform doping effect	1.74×10^{-7}	m
nsub	*Nsub*	Doping concentration	6.0×10^{16}	cm^{-3} Note: T-Spice assigns units of m^{-3} to values > 10^{23}.
pclm	*Pclm*	Channel-length modulation effect coefficient	1.3	—
pdiblc1	*Pdiblc1*	1st output resistance DIBL effect correction parameter	0.39	—
pdiblc2	*Pdiblc2*	2nd output resistance DIBL effect correction parameter	0.0086	—
pdiblcb	*Pdiblcb*	Body effect coefficient of DIBL correction parameters	0.0	V^{-1}
prwb	*Prwb*	Body effect coefficient of **rdsw**	0	$V^{1/2}$
prwg	*Prwg*	Gate bias coefficient of **rdsw**	0	V^{-1}
pscbe1	*Pscbe1*	Substrate current body effect coefficient 1	4.24×10^{8}	V/m
pscbe2	*Pscbe2*	Substrate current body effect coefficient 2	1.0×10^{-5}	V/m
pvag	*Pvag*	*Vg* dependence of *Rout* coefficient	0.0	V
rdsw	*Rdsw*	Source/drain resistance per unit width	0.0	Ω/μm
tox	*Tox*	Gate oxide thickness	1.50×10^{-8}	m
u0	*μ0*	Low-field mobility at **tnom**	670 [n] 250 [p]	$cm^2/V \cdot s$ (≥ 1) $m^2/V \cdot s$ (<1)

Parameter	***Symbol***	***Description***	***Default***	***Units***
ua	*Ua*	Linear *Vgs* dependence of mobility	2.25×10^{-9}	m⁄V
ub	*Ub*	Quadratic *Vgs* dependence of mobility	5.87×10^{-18}	m^2/V^2
uc	*Uc*	Body-bias dependence of mobility	-4.65×10^{-11}	V^{-1}
vbm	*Vbm*	Maximum body voltage	v3.0: –5.0 ≥v3.1: –3.0	V
vfb	vfb	DC flatband voltage.	-1	V
vfbcv	vfbcv	Flatband voltage used in charge/capacitance equations when **vfbflag**=**1** and **capmod**=**0**.	-1	V
vfbflag	vfbflag	Selects vfb for **capmod**=**0**. (Vers. 3.2+)	0	—
voff	*Voff*	Threshold voltage offset	–0.08	V
voffcv	Voffcv	C-V parameter for weak to strong inversion transition	0	—
vsat	ν*sat*	Saturation velocity at **tnom**	8.0×10^4	m⁄s
vtho | **vth0**	*Vth0*	Threshold voltage.	0.7 [n] –0.7 [p]	V
w0	*W0*	Narrow width effect coefficient	2.5×10^{-6}	m
wr	*Wr*	Width offset from *Weff* for *Rds* calculation	1.0	—
xj	*Xj*	Junction depth	1.5×10^{-7}	m

AC and Capacitance Parameters

Parameter	*Symbol*	*Description*	*Default*	*Units*
cf	*Cf*	Fringing field capacitance	Computed,	F/m
cgbo	*Cgbo*	Gate/bulk overlap capacitance per unit channel length	0.0	F/m
cgdl	*Cgdl*	Light doped drain-gate region overlap capacitance	0.0	F/m
cgdo	*Cgdo*	Gate/drain overlap capacitance per unit channel width	0.0	F/m
cgsl	*Cgsl*	Light doped source-gate region overlap capacitance	0.0	F/m
cgso	*Cgso*	Gate/source overlap capacitance per unit channel width	0.0	F/m
ckappa	C_κ	Coefficient for lightly doped region overlap capacitance	0.6	F/m
clc	*CLC*	Constant term for short-channel model	0.1×10^{-6}	m
cle	*CLE*	Exponential term for short-channel model	0.6	—
xpart	*Xpart*	Flag for channel charge partitioning	0	—

Length and Width Parameters

Parameter	*Symbol*	*Description*	*Default*	
dlc	*DLC*	Length offset fitting parameter from C-V	**lint**	m
dwb	*dWb*	Coefficient of *Weff* substrate body bias dependence	0.0	$(m/V)^{1/2}$
dwc	*DWC*	Width offset fitting parameter from C-V	**wint**	m
dwg	*dWg*	Coefficient of *Weff* gate dependence	0.0	m/V
lint	*Lint*	Length offset fitting parameter from I-V without bias	0.0	m
ll	*Ll*	Coefficient of length dependence for length offset	0.0	m^{lln}
llc	Llc	Coefficient of length dependence for C-V channel width offset	**ll**	m^{lln}
lln	*Lln*	Power of length dependence for length offset	1.0	—
lw	*Lw*	Coefficient of width dependence for length offset	0.0	m^{lwn}

Parameter	***Symbol***	***Description***	***Default***	
lwc	Lwc	Coefficient of width dependence for C-V channel length offset	**lw**	m^{lwn}
lwl	*Lwl*	Coefficient of length and width cross terms for length offset	0.0	$m^{lln+lwn}$
lwlc	Lwlc	Coefficient of length and width cross terms for C-V channel length offset	**lwlc**	$m^{lln+lwn}$
lwn	*Lwn*	Power of width dependence for length offset	1.0	—
wint	*Wint*	Width offset fitting parameter from I-V without bias	0.0	m
wl	*Wl*	Coefficient of length dependence for width offset	0.0	m^{wln}
wlc	Wlc	Coefficient of length dependence for C-V channel width offset	**wl**	m^{wln}
wln	*Wln*	Power of length dependence for width offset	1.0	—
ww	*Ww*	Coefficient of width dependence for width offset	0.0	m^{wwn}
wwc	Wwc	Coefficient of width dependence for C-V channel width offset	**ww**	m^{wwn}
wwl	*Wwl*	Coefficient of length and width cross terms for width offset	0.0	$m^{wwn+wln}$
wwlc	Wwlc	Coefficient of length and width cross terms for C-V channel width offset	**wwl**	$m^{wwn+wln}$
wwn	*Wwn*	Power of width dependence for width offset	1.0	—
xl | dl | ldel	*Xl*	Mask and etching length change	0.0	m
xw | dw | wdel |	*Xw*	Mask and etching width change	0.0	m

Temperature Parameters.

Parameter	***Symbol***	***Description***	***Default***	
at	*At*	Temperature coefficient of **vsat**	3.3 × 10^4	m⁄s
CTA | CTC	*CTA | CTC*	Temperature coefficient for *Cj*	0.0	deg^{-1}
CTP	*CTP*	Temperature coefficient for *Cjsw*	0.0	deg^{-1}
EG	*Eg(0)*	Energy gap at 0° K (Si: 1.166, Ge: 0.74, and GaAs: 1.52)	1.16	eV
GAP1	*GAP1*	Coefficient in energy gap temperature equation (Si: 4.73 × 10^{-4}, Ge: 4.77 × 10^{-4}, and GaAs: 5.41 × 10^{-4})	7.02 × 10^{-4}	eV/deg

Parameter	*Symbol*	*Description*	*Default*	
GAP2	*GAP2*	Coefficient in energy gap temperature equation (Si: 636, Ge: 235, and GaAs: 204	1108	deg
kt1	*Kt1*	Temperature coefficient of *Vth*	–0.11	V
kt1l	*Kt1l*	Channel length sensitivity of **kt1**	0.0	V·m
kt2	*Kt2*	Body bias coefficient of **kt1**	0.022	—
prt	*Prt*	Temperature coefficient for **rdsw**	0.0	Ω·µm
PTA	*PTA*	Temperature coefficient for *Pb*	0.0	deg^{-1}
PTP	*PTP*	Temperature coefficient for *Pbsw*	0.0	deg^{-1}
tlev	*tlev*	Temperature equation selector	0	—
tlevc	*tlevc*	Temperature equation selector for junction capacitance and contact potential	0	—
tnom | tref	T_{ref}	Reference temperature	global **tnom** (25.0)	deg
TRD	*TRD*	Temperature coefficient for *Rd*	0.0	deg^{-1}
TRS	*TRS*	Temperature coefficient for *Rs*	0.0	deg^{-1}
ua1	*Ua1*	Temperature coefficient of **ua**	4.31×10^{-9}	m⁄V
ub1	*Ub1*	Temperature coefficient of **ub**	-7.61×10^{-18}	m^2/V^2
uc1	*Uc1*	Temperature coefficient of **uc**	–5.6e-11	V^{-1}
ute	µ*te*	Temperature coefficient of mobility	–1.5	—
XTI	*XTI*	Saturation current temperature exponent	0.0	—

Bin Description Parameters.

Parameter	*Symbol*	*Description*	*Default*
binunit	*binunit*	Bin unit selector	1

Process Parameters

Parameter	*Symbol*	*Description*	*Default*	*Units*
em	*Em*	Maximum electric field	4.1×10^{7}	V⁄m

Parameter	***Symbol***	***Description***	***Default***	***Units***
gamma1	$\gamma 1$	*Vth* coefficient 1	Computed	$V^{1/2}$
gamma2	$\gamma 2$	*Vth* coefficient 2	0.0	$V^{1/2}$
vbx	*Vbx*	*Vth* transition body voltage	Computed	V
xt	*Xt*	Doping depth	1.55×10^{-7}	m

NonQuasi-Static Parameters

Parameter	***Symbol***	***Description***	***Default***
elm	*elm*	Elmore constant	5

Equations

For the complete set of equations describing the Level 49 model, see the *BSIM3v3 Manual* (Ko et al. 1995).

Drain Current

$$I_{ds} = \frac{I_{dso(V_{ds,eff})}}{1 + \frac{R_{ds} I_{dso(V_{ds,eff})}}{V_{ds,eff}}} \left(1 + \frac{V_{ds} - V_{ds,eff}}{V_A}\right)\left(1 + \frac{V_{ds} - V_{ds,eff}}{V_{ASCBE}}\right) \qquad (0.398)$$

where

$$I_{dso} = \frac{W_{eff} \mu_{eff} C_{ox} V_{gst,eff}\left(1 - A_{bulk} \frac{V_{ds,eff}}{2(V_{gst,eff} + 2\nu_t)}\right) V_{ds,eff}}{L_{eff}\left(1 + \frac{V_{ds,eff}}{E_{sat} L_{eff}}\right)} \qquad (0.399)$$

$$V_{ds,eff} = V_{dsat} - \frac{1}{2}(V_{dsat} - V_{ds} - \delta + \sqrt{(V_{dsat} - V_{ds} - \delta)^2 + 4\delta V_{dsat}}) \qquad (0.400)$$

$$E_{sat} = \frac{2\nu_{sat}}{\mu_{eff}} \qquad (0.401)$$

$$\nu_t = \frac{kT}{q} \qquad (0.402)$$

Variables for which equations are not given here are as follows.

V_A Early voltage

V_{ASCBE}	Early voltage due to substrate current-induced body effect
$V_{gst,eff}$	Effective $Vgs - Vth$ (Vth: effective threshold voltage)
μ_{eff}	Effective mobility
A_{bulk}	Bulk charge effect factor
V_{dsat}	Drain saturation voltage

Gate Charge

$$Q_g = -(Q_{acc} + Q_{sub0} + \delta Q_{sub} + Q_{inv}) \qquad (0.403)$$

Variables for which equations are not given here are as follows.

Q_{acc}	Channel majority or accumulation charge
Q_{sub0}	Substrate charge at $Vds = 0$
Q_{sub}	Non-uniform substrate charge in presence of drain bias
Q_{inv}	$Qs + Qd$ = channel minority or inversion charge

MOSFET Levels 14 and 54 (BSIM4 Revision 5)

Parameters

```
.model name nmos|pmos level=14|54 [parameters]
```

Levels 14 and 54 are fully compliant with the original UC Berkeley release of BSIM4 Revision 5. For standard model parameters and equations, refer to the Berkeley manual **BSIM450.pdf**. Specific device instance statements and their parameters are shown in the following section.

Syntax

General MOSFET device parameters (length, width, drain, source, etc.) are described in the device statement chapter under MOSFET (m) on page 164. In the case of device values which have corresponding model values, the device settings override the model settings.

Device instance parameters for BSIM4 are as follows:

```
mname drain gate source bulk model [l=L] [w=W] [ad=Ad] [pd=Pd] [as=As]
    [ps=Ps] [nrd=Nrd] [nrs=Nrs] [M=M] [acnqsmod=acnqsmod] [geomod=geomod]
    [min=min] [nrd=nrd] [nrs=nrs] [rbdb=rbdb] [rbodymod=rbodymod] [rbpb=rbpb]
    [rbpd=rbpd] [rbps=rbps] [rbsb=rbsb] [rgatemod=rgatemod] [rgeomod=rgeomod]
    [trnqsmod=trnqsmod] [sa=sa] [sb=sb] [sd=sd]
```

acnqsmod	AC small-signal NQS model selector
geomod	Geometry-dependent parasitics model selector - specifying how the end S/D diffusions are connected
min	Wheter to minimize the number of drain or source diffusions for even-number finger devices
nf	Number of device fingers
nrd	Number of drain diffusion squares
nrs	Number of source diffusion squares
rbdb	Resistance connected between dbNode and bNode
rbodymod	Substrate resistance network model selector
rbpb	Resistance connected between bNodePrime and bNode
rbpd	Resistance connected between bNodePrime and dbNode
rbps	Resistance connected between bNodePrime and sbNode
rbsb	Resistance connected between sbNode and bNode
rgatemod	Gate resistance model selector
rgeomod	Source/drain diffusion resistance and contact model selector - specifying the end S/D contact type: point, wide or merged, and how S/D parasitics resistance is computed
trnqsmod	Transient NQS model selector
sa	Distance between OD edge to Poly from one side

sb	Distance between OD edge to Poly from other side
sd	Distance between neighboring fingers

Note: Table-based model analysis is not supported for BSIM4.

MOSFET Levels 44 and 55 (EKV Revision 2.6)

Parameters

```
.model name nmos|pmos level=44|55 [parameters]
```

For model parameters, model equations and device instance statements and their parameters, refer to the Swiss Federal Institute of Technology manual **EKV_v262.pdf**.

Note: Table-based model analysis is not supported for EKV.

MOSFET Level 57 (BSIM3SOI)

BSIMSOI is an international standard model for silicon-on-insulator (SOI) circuit design, adjunct to the BSIM3v3 framework.

Parameters

```
.model name nmos|pmos level=57 [parameters]
```

For standard model parameters and equations, refer to the Berkeley manuals **BSIMSOI3p1.pdf** and **BSIMSOI3p2.pdf.** Specific device instance statements and their parameters are shown in the following section.

Syntax

General MOSFET device parameters (length, width, drain, source, etc.) are described in the device statement chapter under MOSFET (m) on page 164. In the case of device values which have corresponding model values, the device settings override the model settings.

Device instance parameters for the BSIMSOI MOSFET model are as follows:

```
mname d g s e [p] [b] [t] model [L=l] [W=w] [P=p] [B=b] [T=t] [AD=ad]
    [AS=as] [PD=pd] [PS=ps] [NRS=nrs] [NRD=nrd] [NRB=nrb] [M=M]
    [OFF] [BJTOFF=bjtoff] [IC=ds gs bs es ps initial voltages] [RTH0=rth0]
    [CTH0=cth0] [NBC=nbc] [NSEG=nseg] [PDBCP=pdbcp] [PSBCP=psbcp]
    [AGBCP=agbcp] [AEBCP=aebcp] [VBSUSR=vbsusr] [TNODEOUT]
    [RGATEMOD=rgatemod] [SOIMOD=soimod] [FRBODY=frbody]
```

d Drain node

g Front gate node

s Source node

e Back gate or substrate node

p Optional external body contact node

b Optional internal body node

t Optional temperature node

ad Drain diffusion area

aebcp Parasitic gate-to-body overlap area for body contact

agbcp Parasitic perimeter length for body contact at drain side

as Source diffusion area

bjtoff Turn off BJT current if equal to 1

cth0 Thermal capacitance per unit width:

- if not specified, CTH0 is extracted from model card
- f specified, it will override the one in model card

frbody Layout-dependent body resistance coefficient

ic	Initial guess in the order of (Vds, Vgs, Vbs, Ves, Vps). (Vps will be ignored in the case of a 4-terminal device.)
l	Channel length
nbc	Number of body contact isolation edge
nrb	Number of squares in body series resistance
nrd	Number of squares in drain series resistance
nrs	Number of squares in source series resistance
nseg	Number of segments for channel width partitioning. (The effective channel width may change due to the body contact, please refer to the equations in **BSIMSOI3p1.pdf** and **BSIMSOI3p2.pdf**.
off	Device initial conditions off in DC operating point calculations
pd	Drain diffusion perimeter area
pdbcp	Parasitic perimeter length for body contact at drain side
ps	Drain diffusion perimeter length
psbcp	Parasitic perimeter length for body contact at source side
rgatemod	Gate resistance model selector
rsh	Number of squares in drain series resistance
rth0	Thermal resistance per unit width: if not specified, RTH0 is extracted from model card if specified, it will override the one in model card
soimod	SOI model selector for PD/FD operation ▪ Soimod=0: BSIMPD (partially depleted) ▪ Soimod=1: unified model for PD&FD ▪ Soimod=2: ideal FD (fully depleted)
tnodeout	Flag indicating external temperature node
vbsusr	Optional initial value of Vbs specified by user for transient analysis
w	Channel width

SOI Modes

There are three modes in BSIMSOI, soimod = 0, 1 or 2. BSIMPD (soimod = 0) can be used to model the PD SOI device, where the body potential is independent of DVbi (VBS > DVbi). Therefore the calculation of DVbi is skipped in this mode. On the other hand, the ideal FD model (soimod = 2) is for the FD device with body potential equal to DVbi. Hence the calculation of body current/charge, which is essential to the PD model, is skipped. For the unified SOI model (soimod = 1), however, both DVbi and body current/charge are calculated to capture the floating-body behavior exhibited in FD devices.

This unified model covers both BSIMPD and the ideal FD model.

Body and Temperature Nodes

There are three optional nodes, P, B and T nodes. Nodes P and B are used for body contact devices. If the TNODEOUT flag is not set, when you specify four nodes, this element is a four terminal device, i.e., floating body. If you specify five nodes, the fifth node represents the external body contact node (P). There is a body resistance between the internal body node and the P node. In both these cases, an internal body node is created, but it is not accessible in the circuit deck. However, if you specify six nodes, the fifth node will represent the P node and the sixth node will represent the internal body node (B). This configuration is useful for distributed body resistance simulation.

If the TNODEOUT flag is set, the last node is interpreted as the temperature node. In this case, when you specify five nodes, it is a floating body. If you specify six nodes, it is a body-contacted case. Finally, if you specify seven nodes, it is a body-contacted case with an accessible internal body node. The temperature node is useful for thermal coupling simulation.

Note: Table-based model analysis is not supported for BSIMSOI.

MOSFET Levels 9 and 50 (Philips MOS 9)

Philips MOS Model 9 is a compact MOS-transistor model, intended for the simulation of circuit behaviour with emphasis on analog applications. The model gives a complete description of all transistor-action related quantities: nodal currents and charges, noise-power spectral densities and weak-avalanche currents. The equations describing these quantities are based on the gradual-channel approximation with a number of first-order corrections for small-size effects. The consistency is maintained by using the same carrier-density and electrical-field expressions in the calculation of all model quantities. MOS Model 9 only provides a model for the intrinsic transistor. Junction charges and leakage currents are not included. They are covered by the separate **Juncap** model.

The MOS 9 model is fully documented in **Philips MOS 9.**

For further detailed information about the MOS 9 model, please refer to the Philips Compact Model Webpage:

📖 http://www.semiconductors.philips.com/Philips_Models/mos_models/model9

Parameters

The MOS 9 model uses the following syntax.

```
.model name nmos|pmos level=[9|50] | model=modelname [parameters]
```

T-Spice includes support for MOS 9 versions 902 and 903, and for both geometrical and electrical based model parameter sets.

The available *modelname* values for the MOS 9 model selection are:

Modelname	*Description*
mos902	Mos 9 level 902, geometrical
mos902e	Mos 9 level 902, electrical
mos903 (default)	Mos 9 level 903, geometrical
mos903e	Mos 9 level 903, electrical

MOSFET Levels 11 and 63 (Philips MOS 11)

MOS Model 11 has been developed as the successor of MOS Model 9. It is a symmetrical, surface-potential-based model, giving an accurate physical description of the transition from weak to strong inversion. MOS 11 includes an accurate description of all physical effects important for modern and future CMOS technologies, such as:

mobility reduction
bias-dependent series resistance
velocity saturation
conductance effects (CLM, DIBL, etc.)
gate leakage current
gate-induced drain leakage
gate depletion
quantum-mechanical effects
bias-dependent overlap capacitances

The description of the source-bulk and drain-bulk junction diode is not included in MOS Model 11. The behaviour of these junction diodes is modelled by the Juncap model. This model has to be added between the source and bulk node and between the drain and bulk node. The MOS 11 model is fully documented in Philips MOS 11.

The MOS 11 model is fully documented in **Philips MOS 11.**

For further detailed information about the MOS 11 model, please refer to the Philips Compact Model Webpage:

📖 http://www.semiconductors.philips.com/Philips_Models/mos_models/model11

Parameters

The MOS 11 model uses the following syntax.

```
.model name nmos|pmos level=[11|63] | model=modelname [parameters]
```

T-Spice includes support for Mos 11 versions 1100, 1101, and 1102. Each of these versions, in turn, offers a selection of electrical or geometrical based parameterization, modeling of self-heating effects, and model binning

The available *modelname* values for the Mos 11 model are:

Modelname	*Description*
mos1100	Mos 11 level 1100, geometrical
mos1100e	Mos 11 level 1100, electrical
mos1101	Mos 11 level 1101, electrical
mos1101t	Mos 11 level 1101, electrical, self-heating
mos11010	Mos 11 level 1101, geometrical
mos11010t	Mos 11 level 1101, geometrical, self-heating
mos11011	Mos 11 level 1101, geometrical, binning

mos11011t	Mos 11 level 1101, geometrical, binning, self-heating
mos1102	Mos 11 level 1102, electrical
mos1102t	Mos 11 level 1102, electrical, self-heating
mos11020 (default)	Mos 11 level 1102, geometrical
mos11020t	Mos 11 level 1102, geometrical, self-heating
mos11021	Mos 11 level 1102, geometrical, binning
mos11021t	Mos 11 level 1102, geometrical, binning, self-heating

MOSFET Level 20 (Philips MOS 20)

MOS Model 20 is a compact LDMOS model, which combines the MOSFET operation of the channel region with that of the drift region under the thin gate oxide. As such, it is aimed as a successor of MOS Model 9 in series with MOS Model 31. MOS Model 20 has especially been developed to improve the convergence behaviour during simulation, by having the voltage at the transition from the channel region to the drift region calculated inside the model itself.

The MOS 20 model is fully documented in **Philips MOS 20.**

For further detailed information about the MOS 20model, please refer to the Philips Compact Model Webpage:

http://www.semiconductors.philips.com/Philips_Models/high_voltage/model20

Parameters

The MOS 20 model uses the following syntax.

```
.model name nmos|pmos level=20 | model=modelname [parameters]
```

T-Spice includes support for MOS 20 version 2001 with geometrical and electrical model parameterization, and self-heating effects.

The available *modelname* values for the MOS 20 model selection are:

Modelname	***Description***
mos2001 (default)	Mos 20 level 2001, geometrical
mos2001e	Mos 20 level 2001, electrical
mos2001et	Mos 20 level 2001, electrical, self-heating

MOSFET Level 30 (Philips MOS 30)

MOS Model 30 is a long channel JFET/MOSFET model developed to describe the drift region of LDMOS, EPMOS and VDMOS devices.

Note: MOS Model 30 has been replaced with the MOS 31 model, and is provided for historical and compatibility purposes only. It's use is not recommended.

The MOS 30 model is fully documented in **Philips MOS 30.**

Parameters

The MOS 30 model uses the following syntax.

```
.model name nmos|pmos level=30 | model=mos3002 [parameters]
```

T-Spice includes support for MOS 30 version 3002 with electrical model parameterization.

MOSFET Level 31 (Philips MOS 31)

MOS Model 31 is a physics based transistor model to be used in circuit simulation and IC-design of analog high-voltage applications. The model describes the electrical behaviour of a junction-isolated accumulation/depletion-type MOSFET. The model is used as the drain extension of high-voltage MOS devices, like the Lateral Double-diffused MOS (LDMOS), the Vertical Double-diffused MOS (VDMOS), and the Extended MOS transistors. Physical effects included in MOS Model 31:

Both accumulation and depletion underneath the gate oxide;
Depletion from the substrate (a pn-junction);
Pinch-off effects;
Velocity saturation; and
Temperature scaling.

The MOS 31 model is fully documented in **Philips MOS 31.**

For further detailed information about the MOS 31 model, please refer to the Philips Compact Model Webpage:

🕮 http://www.semiconductors.philips.com/Philips_Models/high_voltage/model31

Parameters

The MOS 31 model uses the following syntax.

```
.model name nmos|pmos level=31 | model=modelname [parameters]
```

T-Spice includes support for MOS 31 version 3100 with and without self-heating effects.

The available *modelname* values for the MOS 31 model selection are:

Modelname	***Description***
mos3100 (default)	Mos 31 level 3100, electrical
mos3100t	Mos 31 level 3100, electrical, self-heating

MOSFET Level 40 (Philips MOS 40)

MOS Model 40 is a physics based transistor model to be used in circuit simulation and IC-design of analogue high-voltage applications processed in Silicon-on-Insulator (SOI). The model describes the electrical behavior of an accumulation/depletion-type MOSFET in SOI. The model is used as drain extension of high-voltage MOS devices, like the Lateral Double-diffused MOS (LDMOS), the Vertical Double-diffused MOS (VDMOS), and the Extended MOS transistors.
Physical effects in MOS Model 40 include:

Both accumulation and depletion underneath the gate oxide;
Both accumulation and depletion from the substrate (an oxide layer);
Pinch-off effects;
Velocity saturation; and
Temperature scaling.

The MOS 40 model is fully documented in **Philips MOS 40.**

For further detailed information about the MOS 40 model, please refer to the Philips Compact Model Webpage:

🕮 http://www.semiconductors.philips.com/Philips_Models/high_voltage/model40

Parameters

The MOS 40 model uses the following syntax.

```
.model name nmos|pmos level=40 | model=modelname [parameters]
```

T-Spice includes support for MOS 40 version 40 with and without self-heating effects.

The available *modelname* values for the MOS 40 model selection are:

Modelname	***Description***
mos40 (default)	Mos 40 level 40, electrical
mos40t	Mos 40 level 40, electrical, self-heating

MOSFET Level 100 (Penn State & Philips PSP Model)

The PSP model is a new compact MOSFET model, which has been jointly developed by Philips Research and Penn State University, and is able to accurately model present-day and upcoming deep-submicron bulk CMOS technologies.

The PSP model is a symmetrical, surface-potential-based model, giving an accurate physical description of the transition from weak to strong inversion. The PSP model includes an accurate description of all physical effects important for modern and future CMOS technologies, such as:

mobility reduction
bias-dependent series resistance
velocity saturation
conductance effects (CLM, DIBL, etc.)
lateral doping gradient effect
mechanical stress related to STI
gate leakage current
gate-induced drain leakage
gate depletion
quantum-mechanical effects bias-dependent overlap capacitances

In addition, it gives an accurate description of charges and currents and their first-order derivatives (transconductance, conductance, capacitances), but also of their higher-order derivatives. In other words, it gives an accurate description of MOSFET distortion behaviour, and as such the PSP model is suitable for digital, analog as well as RF circuit design.

The PSP model is fully documented in **Penn State & Philips PSP Model.**

For further detailed information about the PSP model, please refer to the Philips Compact Model Webpage:

📖 http://www.semiconductors.philips.com/Philips_Models/high_voltage/model40

Parameters

The PSP model uses the following syntax.

```
.model name nmos|pmos level=100|1000 | model=modelname [parameters]
```

T-Spice includes support for PSP version 100.1 with both electrical and geometrical model parameterization.

The available *modelname* values for the PSP model selection are:

Modelname	***Description***
psp100 or psp100e (default)	PSP model version 100.1, electrical
psp1000 or psp100g	PSP model version 100.1, geometrical

Additional MOSFET Parameters

This section describes additional parameters, including parasitics, used by the equations describing MOSFET levels 1-3 and BSIM (levels 1, 2, 3, 14, 28, 47, 49, 53, and 54).

Parameters

Parameter	*Symbol*	*Description*	*Default*	*Units*
acm	*ACM*	Source/drain area calculation method	0	—
cj | **cjm**	*Cj*	Source/drain bottom junction capacitance	5.7911×10^{-4} [level 49] 5.0×10^{-4} [else]	F⁄m^2
cjgate	*Cj,gate*	Zero-bias gate edge sidewall junction capacitance	0.0	F⁄m
cjsw | **cjw**	*Cjsw*	Source/drain sidewall junction capacitance	0.0	F⁄m
expli	EXPLI	Current limit	1.0×10^{15}	A
hdif	*Hdif*	Length of heavily doped diffusion from contact to lightly doped region	0.0	m
is	*Is*	Bulk saturation current	0.0 [level 49] 1.0×10^{-14} [else]	A
js | **ijs**	*Js*	Source/drain junction reverse saturation current density	0.0	A⁄m^2
jsw | **jssw**	*Jsw*	Source/drain sidewall junction reverse saturation current density	0.0	A⁄m
ld | **dlat** | **latd**	*Ld*	Lateral diffusion into channel from source/drain diffusion	0.75 · **xj**	—
ldif	*Ldif*	Length of lightly doped diffusion adjacent to gate	0.0	m
lmax	*Lmax*	Maximum channel length	0.0	m
lmin	*Lmin*	Minimum channel length	0.0	m
meto	*Meto*	Fringing field factor for overlap capacitance calculation	0.0	m
mj | **mj0**	*Mj*	Source/drain bottom junction capacitance grading coefficient	0.5	—
mjsw | **mjw**	*Mjsw*	Source/drain sidewall junction capacitance grading coefficient	0.33	—
n	*N*	Emission coefficient	1.0	—
nds	*Nds*	Reverse bias slope coefficient	1.0	—
pb | **pj**	*Pb*	Source/drain junction built-in potential	0.8 [level 49] 1.0 [else]	V
pbsw | **pjw** | **php**	*Pbsw*	Source/drain sidewall junction capacitance built-in potential	**pb**	V

Parameter	*Symbol*	*Description*	*Default*	*Units*
prdt	*Prdt*	Drain resistance temperature coefficient	0.0	—
prst	*Prst*	Source resistance temperature coefficient	0.0	—
rd	*Rd*	Drain resistance (**rsh** override)	0.0	Ω
rdc	*Rdc*	Drain contact resistance	0.0	Ω
rs	*Rs*	Source resistance (**rsh** override)	0.0	Ω
rsc	*Rsc*	Source contact resistance	0.0	Ω
rsh | **rshm**	*Rsh*	Source/drain sheet resistance	0.0	Ω/□
vnds	*Vn,ds*	Reverse diode current transition point	–1	V
wmax	*Wmax*	Maximum channel width	1.0	m
wmin	*Wmin*	Minimum channel width	0.0	m
wmlt	Wmlt	Width diffusion layer shrink reduction factor	1.0	—
xlref	*Xl,ref*	Difference between physical and drawn channel length	0.0	m
xwref	*Xw,ref*	Difference between physical and drawn channel width	0.0	m

Equations

Parasitic Resistances

MOSFET parasitics are simulated as resistances in series with the source ($R_{s_{eff}}$) and drain ($R_{d_{eff}}$). How these resistances are computed depends on the area calculation method (**acm**) specified.

acm	**rseff**	**rdeff**
0, **10**		
	$R_{s_{eff}} = N_{rs} \cdot R_{sh} + R_{sc} \quad \text{if } N_{rs} \cdot R_{sh} \neq 0$ $R_{s_{eff}} = R_s + R_{sc} \quad \text{otherwise}$	$R_{d_{eff}} = N_{rd} \cdot R_{sh} + R_{dc} \quad \text{if } N_{rd} \cdot R_{sh} \neq 0$ $R_{d_{eff}} = R_s + R_{dc} \quad \text{otherwise}$

acm	**rseff**	**rdeff**
1, **11**	$R_{s_{eff}} = \frac{(L_d + L_{dif}) \cdot R_s}{W_{eff}} + N_{rs} \cdot R_{sh} + R_{sc}$	$R_{d_{eff}} = \frac{(L_d + L_{dif}) \cdot R_d}{W_{eff}} + N_{rd} \cdot R_{sh} + R_{dc}$
2, **3**, **12**, **13**	If N_{rs} is specified: $R_{s_{eff}} = \frac{(L_d + L_{dif}) \cdot R_s}{W_{eff}} + N_{rs} \cdot R_{sh} + R_{sc}$ Otherwise: $R_{s_{eff}} = \frac{(L_d + L_{dif}) \cdot R_s + R_{sh} \cdot H_{dif_{eff}}}{W_{eff}}$	If N_{rd} is specified: $R_{d_{eff}} = \frac{(L_d + L_{dif}) \cdot R_d}{W_{eff}} + N_{rd} \cdot R_{sh} + R_{dc}$ Otherwise: $R_{d_{eff}} = \frac{(L_d + L_{dif}) \cdot R_d + R_{sh} \cdot H_{dif_{eff}}}{W_{eff}}$

Parasitic Diodes

Parasitic diodes are added for each MOSFET that uses direct model evaluation, and for all MOSFETs if the **mosparasitics** option is on.

One diode is placed between the MOSFET's bulk and source terminals, and the other between the bulk and drain.

Diode characteristics are determined by the device parameters **as**, **ad**, **pd**, **ps**, and **geo**, as well as the model parameters **acm**, **cj**, **cjsw**, **cjgate**, **js**, **jsw**, **is**, **n**, **nds**, **vnds**, and **hdif**. The quantity **weff** also plays a role in determining default values for source and drain areas and perimeters for some values of **acm**.

If the MOSFET bulk-source voltage **vbs** is positive (the bulk-source diode is forward biased), then the bulk-source DC current **ibs** is

$$\textbf{ibs} = \textbf{isatbs} \cdot \exp(\textbf{vbs} / (\textbf{n} \cdot \textbf{vt}) - 1) \tag{0.404}$$

where **vt** = kT/q (the thermal voltage), and **isatbs** is the saturation current:

$$\textbf{isatbs} = \textbf{js} \cdot \textbf{aseff} + \textbf{jsw} \cdot \textbf{pseff} \tag{0.405}$$

aseff and **pseff** are described below.
If this computed value of isatbs is zero, then isatbs will be set to the **is** parameter value.

If the MOSFET bulk-drain voltage **vds** is positive (the bulk-drain diode is forward biased), then the bulk-drain DC current is

$$\mathbf{ibd} = \mathbf{isatbd} \cdot \exp(\mathbf{vbd} / (\mathbf{n} \cdot \mathbf{vt}) - 1) \qquad (0.406)$$

where **isatbd** is the saturation current:

$$\mathbf{isatbd} = \mathbf{js} \cdot \mathbf{adeff} + \mathbf{jsw} \cdot \mathbf{pdeff} \qquad (0.407)$$

adeff and **pdeff** are described below.
If this computed value of isatbd is zero, then isatbd will be set to the **is** parameter value.

The exponential function in both diodes is replaced by a linear extension when the current is larger than the value of **expli**. The linear extension is chosen such that the diode current function is continuously differentiable at the transition point where the diode current equals **expli**.

When a MOSFET parasitic diode with saturation current **isat** is reverse-biased with a negative voltage **vdi**, then its current **idi** behaves as follows.

If 0 > **vdi** > **vnds**, then

$$\mathbf{idi} = \mathbf{isat} \cdot \mathbf{vdi} \qquad (0.408)$$

If **vdi** < **vnds**, then

$$\mathbf{idi} = \mathbf{isat} \cdot (\mathbf{vnds} + (\mathbf{vdi} - \mathbf{vnds}) / \mathbf{nds}) \qquad (0.409)$$

Effective Areas and Perimeters

Effective source and drain areas and perimeters depend on the value of parameter **acm** (and, if **acm** = 3, on parameter **geo**). The parameter names are:

aseff	Effective source area
adeff	Effective drain area
pseff	Effective source perimeter
pdeff	Effective drain perimeter

The values of **geo** are:

0	Drain and source not shared by other devices (default)
1	Drain shared with another device
2	Source shared with another device
3	Drain and source shared with other devices

The area and perimeter values are computed as follows. The first table lists the equations used when **acm=0**, **1**, **2**, or **3**. The second table lists equations for **acm=10**, **11**, **12**, or **13**. The parameter **CALCACM** can only be invoked when **acm=12**. The values of **defas**, **defad**, and **moscap** are specified with the **.options** command.

	acm=**0** or **10** with **as**	**acm=0** without **as**		**acm=10** without **as**
$A_{s_{eff}}$	$A_s\cdot(W_{mlt})^2$	l·w defas	if **moscap=1** otherwise	0
$A_{d_{eff}}$	$A_d\cdot(W_{mlt})^2$	$l\cdot w$ defad	if **moscap=1** otherwise	0
$P_{s_{eff}}$	$P_s\cdot W_{mlt}$	$2\cdot(l+w)$ 0	if **moscap=1** otherwise	0
$P_{d_{eff}}$	$P_d\cdot W_{mlt}$	$2\cdot(l+w)$ 0	if **moscap=1** otherwise	0

For acm=1 or 11:

	acm=**1** or **11** with **as**	**acm=1** without **as**	**acm=11** without **as**
$A_{s_{eff}}$	$W_{eff}\cdot W_{mlt}$	$W_{eff}\cdot W_{mlt}$	$W_{eff}\cdot W_{mlt}$
$A_{d_{eff}}$	$W_{eff}\cdot W_{mlt}$	$W_{eff}\cdot W_{mlt}$	$W_{eff}\cdot W_{mlt}$
$P_{s_{eff}}$	$\mathrm{W_{eff}}$	$\mathrm{W_{eff}}$	$\mathrm{W_{eff}}$
$P_{d_{eff}}$	$\mathrm{W_{eff}}$	$\mathrm{W_{eff}}$	$\mathrm{W_{eff}}$

For acm=2 or 12:

	acm=**2** or **12** with **as**	**acm=2** without **as**	**acm=12** without **as**	
$A_{s_{eff}}$	$A_s\cdot(W_{mlt})^2$	$2\cdot H_{dif_{eff}}\cdot W_{eff}$	$2\cdot H_{dif_{eff}}\cdot W_{eff}$ 0	if **calcacm=1** otherwise
$A_{d_{eff}}$	$A_d\cdot(W_{mlt})^2$	$2\cdot H_{dif_{eff}}\cdot W_{eff}$	$2\cdot H_{dif_{eff}}\cdot W_{eff}$ 0	if **calcacm=1** otherwise
$P_{s_{eff}}$	$P_s\cdot W_{mlt}$	$4\cdot H_{dif_{eff}}+2\cdot W_{eff}$	$4\cdot H_{dif_{eff}}+2\cdot W_{eff}$ 0	if **calcacm=1** otherwise
$P_{d_{eff}}$	$P_d\cdot W_{mlt}$	$4\cdot H_{dif_{eff}}+2\cdot W_{eff}$	$4\cdot H_{dif_{eff}}+2\cdot W_{eff}$ 0	if **calcacm=1** otherwise

For acm=3 or 13:

	acm=**3** or **13** with **as**	**acm=3** without **as**		**acm=13** without **as**
$A_{s_{eff}}$	$A_s\cdot(W_{mlt})^2$	$2\cdot H_{dif_{eff}}\cdot W_{eff}$ $H_{dif_{eff}}\cdot W_{eff}$	if **geo=0** or **1** otherwise	0

	acm=**3** or **13**	**acm**=**3**		**acm**=**13**
	with **as**	without **as**		without **as**
$A_{d_{eff}}$	$A_d{\cdot}(W_{mlt})^2$	$2 \cdot H_{dif_{eff}} \cdot W_{eff}$	if **geo**=**0 or 1**	0
		$H_{dif_{eff}} \cdot W_{eff}$	otherwise	
$P_{s_{eff}}$	$P_s{\cdot}W_{mlt}$	$4 \cdot H_{dif_{eff}} + W_{eff}$	if **geo**=**0 or 1**	0
		$2 \cdot H_{dif_{eff}} + W_{eff}$	otherwise	
$P_{d_{eff}}$	$P_d{\cdot}W_{mlt}$	$4 \cdot H_{dif_{eff}} + W_{eff}$	if **geo**=**0 or 1**	0
		$2 \cdot H_{dif_{eff}} + W_{eff}$	otherwise	

Resistor

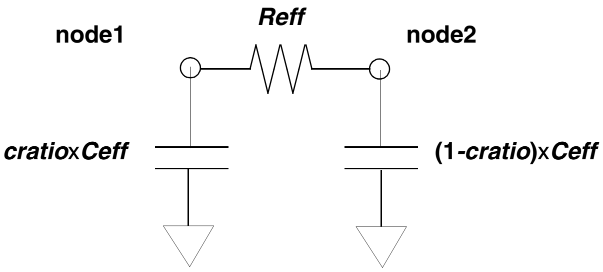
Parameters

```
.model name r [parameter=X]
```

Parameter	*Description*	*Default*	*Units*
bulk	Name of the node used as the bulk node for capacitance.	Gnd	
cap	Default capacitance.	0	F
capsw	Sidewall fringing capacitance.	0	F/m
cratio	Specifies how capacitance is distributed between input and output nodes. The capacitor between ***node1*** and the bulk node has the value **cratio**×***Ceff***, while the capacitor between node2 and the bulk node has the value (1-**cratio**)×***Ceff***. ***Ceff*** is the effective capacitance as described below.	0.5	
cox	Bottomwall capacitance.	0	F/m
di	Relative dielectric constant.	0	
dlr	Difference between the drawn resistor length and its actual length. Multiplied by **.options scalm**. For further information, see **.options** (page 95).	0	m
dw	Difference between the drawn resistor width and its actual width. Multiplied by **.options scalm**. For further information, see **.options** (page 95).	0	m
l	Default length. Multiplied by **shrink** and **.options scalm** to obtain the scaled length. For further information, see **.options** (page 95).	0	m
level	Model selector—not used.		
noise	Default noise multiplier.	1	
rac	Default AC resistance.	DC resistance	Ohm
res	Default resistance.	0	Ohm
rsh	Sheet resistance per square.	0	
shrink	Shrink factor.	1	
tc1c	First-order temperature coefficient for capacitance.	0	deg^{-1}
tc2c	Second-order temperature coefficient for capacitance.	0	deg^{-2}
tc1r	First-order temperature coefficient for resistance.	0	deg^{-1}

Parameter	*Description*	*Default*	*Units*
tc2r	Second-order temperature coefficient for resistance.	0	deg^{-2}
thick	Dielectric thickness.	0	m
tnom | tref	Reference temperature for temperature compensation.	global **tnom** (25.0)	deg
w	Default width. Multiplied by shrink and **.options scalm** to obtain the scaled width. For further information, see **.options** (page 95).	0	m

Large-Signal Model



Equations

Resistance

Effective length is calculated as

$$L_{eff} = L_{scaled} - (2 \cdot dlr \cdot scalm) \tag{0.410}$$

Effective width is calculated as

$$W_{eff} = W_{scaled} - (2 \cdot dw \cdot scalm) \tag{0.411}$$

If element resistance is specified, the effective resistance is

$$R_{eff} = resis\tan ce \times devscale \tag{0.412}$$

Otherwise, if $W_{eff} \cdot L_{eff} \cdot rsh = 0$, then $R_{eff} = res \cdot devscale$.

Or if $W_{eff} \cdot L_{eff} \cdot rsh \neq 0$, then $R_{eff} = L_{eff} \cdot rsh \cdot (devscale) / W_{eff}$.

In AC analysis, the device’s resistance is

$$RAC_{eff} = acres \cdot devscale \tag{0.413}$$

if ***acres*** is given.

$$RAC_{eff} = rac \cdot devscale \tag{0.414}$$

if ***acres*** is not given and model parameter **rac** is given.

$$RAC_{eff} = R_{eff} \tag{0.415}$$

if neither ***acres*** nor **rac** is given.

Note: If T-Spice calculates the effective resistance (R_{eff} or RAC_{eff}) to be less than 10^{-5} Ω, then a warning message is issued and the effective resistance is automatically assigned a value of 10^{-5} Ω.

For additional information on ***devscale***, see the device statement Resistor (r) on page 169.

Capacitance

Effective length is calculated as

$$L_{eff} = L_{scaled} - (2 \cdot dlr \cdot scalm) \tag{0.416}$$

Effective width is calculated as

$$W_{eff} = W_{scaled} - (2 \cdot dw \cdot scalm) \tag{0.417}$$

If element capacitance is specified, the effective capacitance is

$$C_{eff} = c \times devscale \tag{0.418}$$

If **cap** is the only model parameter specified

$$C_{eff} = cap \cdot devscale \tag{0.419}$$

Otherwise,

$$C_{eff} = devscale \cdot (Leff \cdot Weff \cdot cox + 2(L_{eff} + W_{eff}) \times capsw) \tag{0.420}$$

If **cox** is not specified, it is computed in one of two ways.

$$\text{If } \textbf{di} \text{ is specified: } cox = di \cdot \frac{8.8542149 \cdot 10^{-12}}{thick} \tag{0.421}$$

$$\text{If If } \textbf{di} \text{ is not specified: } cox = \frac{3.453148 \cdot 10^{-11}}{thick} \tag{0.422}$$

Note: If ***c***, **cap**, **cox**, and **thick** are not specified, no capacitor is created.

Switch

A current- or voltage-controlled switch.

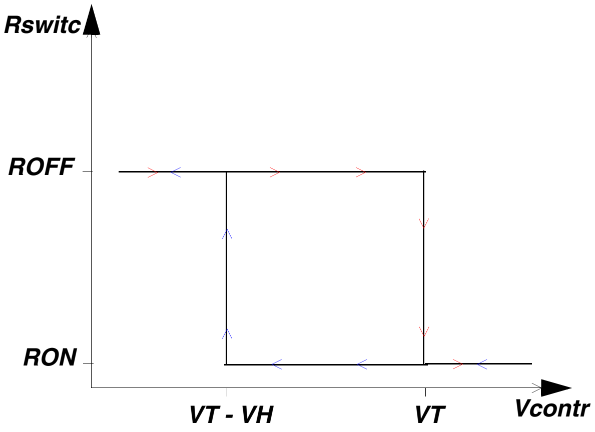
Syntax

```
.model modelname sw|csw [parameter=value [parameter=value [...]]]
```

Name	*Model*	*Description*	*Default*	*Unit*
vt	sw	Threshold voltage	0	V
it	csw	Threshold current	0	A
vh	sw	Hysteresis voltage	0	V
ih	csw	Hysteresis current	0	A
ron	sw, csw	ON resistance	1	Ohm
roff	sw, csw	OFF resistance	1e12	Ohm
dv	sw	Threshold transition width	0.01	V
di	csw	Threshold transition width	1e-8	A
hdt	sw, csw	Threshold transition time	1e-10	s

The T-Spice switch is essentially a controlled resistor. The resistance is ***ron*** when the switch is on, and ***roff*** when the switch is off. The switch changes state between on and off when the controlling voltage or current is at its threshold value.

The following figure shows the resistance of the switch as a function of the controlling variable:



The T-Spice switch elements can display hysteresis, so that the threshold value is different when the control voltage/current is increasing than when it is decreasing. For a voltage-controlled switch, the threshold voltage is ***vt*** when v(*control1, control2*) is increasing, and ***vt-vh*** when v(*control1*, *control2*) is decreasing. For a current-controlled switch, the threshold current is ***it*** when i(*vsource_name*) is increasing, and ***it-ih*** when i(*vsource_name*) is decreasing. The switch is on when the control voltage or current is greater than the threshold value.

The ***dv*** and ***di*** parameters define a small interval around the threshold in which a smooth transition between ***ron*** and ***roff*** is made.

Examples

The following example creates a voltage-controlled switch:

```
.model swmod sw vt=0.7 dv=0.1
```

The following example creates a current-controlled switch:

```
.model swmod csw it=0.7 di=0.1
```

Transmission Line

T-Spice supports two transmission line models:

- *Lossless* line, defined by characteristic impedance and delay. This model is described by Branin 1967.
- *Lossy* line, defined by RLCG parameters. This model is described here.

Equations

The lossy transmission line is modeled with equivalent circuits consisting of cascaded cells or *lumps*, typically comprising discrete resistors, inductors, and capacitors. Lumps essentially discretize the transmission line wave equations over the length of the line. Distributed RLCG values are converted to non-distributed (lumped element) values:

$$Rlump = R \cdot l/n \tag{0.423}$$

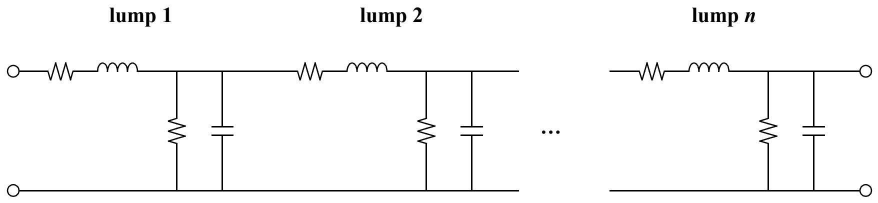
$$Llump = L \cdot l/n \tag{0.424}$$

$$Clump = C \cdot l/n \tag{0.425}$$

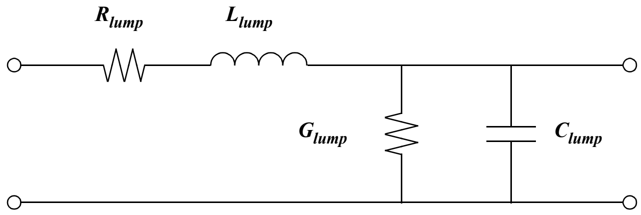
$$Glump = G \cdot l/n \tag{0.426}$$

where *l* is the physical length of the transmission line and *n* is the number of lumps.

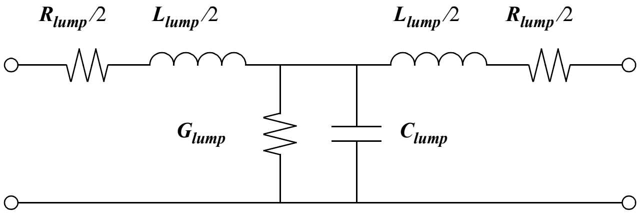
Typically, many lumps are needed to model a transmission line accurately. The number of lumps is specified by the **lumps** parameter on the device statement. A cascade ladder network—the *iterative ladder circuit*—is constructed with the specified number of lumps.



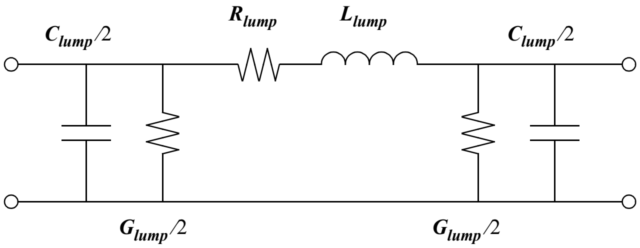
The “gamma” lump type is the most common implementation.



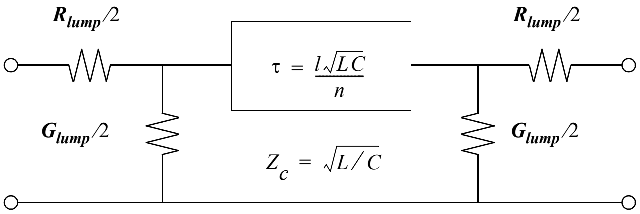
The “tee” lump type is a symmetrical alternative that includes an additional series resistance and inductance per lump.



The “pi” lump type is formed by adding identical shunt elements at the input and output of each lump.



Low-loss transmission lines need not be modeled with RLCG lumps. An alternative approach is to use lossless transmission line sections separated by lumped resistances and conductances. The “hybrid RGT” lump type consists of an ideal transmission line section with series resistances and shunt conductances at input and output. This equivalent circuit defaults to the lossless transmission line when *R* and *G* values are zero, and is therefore the default **lumptype** option.



9 Small-Signal and Noise Models

Introduction

Small-Signal Models

The linear small-signal models for diodes, BJTs, JFETs, MESFETs, and MOSFETs described in this chapter are derived from T-Spice's nonlinear device representations.

Instead of using simplified equations to compute the small-signal model parameters, which can introduce errors at low and high frequencies, T-Spice generates linearized small-signal models directly from the device equations, with the most accurate small-signal results available.

Small-signal parameters are evaluated at the DC operating point. The currents and voltages appearing in the equations in this chapter are all DC values. In addition, the voltages are measured at the "intrinsic" terminals, *inside* the parasitic resistances connected to the external terminals.

Small-signal data are available with the **.acmodel** (page 54) command.

Noise Models

The following parameters are used by many of T-Spice's noise models.

Parameter	*Symbol*	*Description*	*Default*	*Unit*
noiselevel | nlev	—	Noise equation selector (JFET, MESFET, MOSFET only)	2	—
gdsnoise | gdsnoi	*GDSn*	Channel noise coefficient (JFET, MESFET, MOSFET only)	1.0	—
af	*Af*	Flicker noise exponent	1.0	—
kf	*Kf*	Flicker noise coefficient	0.0	—

Noise models add RMS noise current sources to small signal models in order to simulate noise power densities (NPDs) in the device. The NPDs are calculated from model parameter values and DC operating point conditions.

For example, each of the resistors in a circuit, including the parasitic resistances in models, generates thermal noise, whose NPD is inversely proportional to the resistance:

$$\overline{|i_{th}|^2} = \frac{4kT}{R} \cdot \Delta f \tag{0.427}$$

where *k* is Boltzmann's constant, *T* is the temperature in Kelvins, *R* is the resistance, and Δf is the width of the frequency band over which the noise is measured (1 Hz in T-Spice). The corresponding noise current source has a magnitude of $\sqrt{\overline{|i_{th}|^2}}$ and units of $A/\sqrt{Hz}$, and is placed in parallel with the resistor.

The following sections show the NPD calculation for intrinsic noise sources in the models. These sources represent shot and/or flicker noise.

Shot noise is typically computed as

$$\overline{|i_{shot}|^2} = 2q|I_x| \cdot \Delta f \tag{0.428}$$

where q is the electron charge and *Ix* is the DC current into terminal x at the operating point of the device.

Flicker noise is usually modeled by

$$\overline{|i_{flicker}|^2} = \frac{K_f|I_x|^{A_f}}{f} \cdot \Delta f \tag{0.429}$$

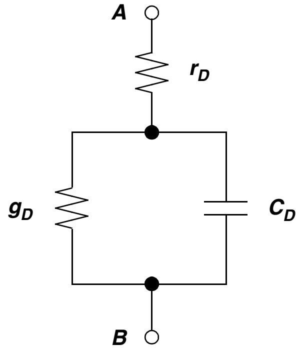
where f is the center frequency of the band over which the noise is being measured.

Where appropriate, alternate formulations for shot and flicker noise are presented for individual models.

The noise source corresponding to the NPD, $\sqrt{\overline{|i_{XY_n}|^2}}$, is connected between the intrinsic X and Y nodes in the noise model.

Diode

Small-Signal Model



The conductance *gD* is the partial derivative of the forward current *ID* with respect to the voltage *VD* across the intrinsic diode.

$$g_D = \left.\frac{\partial I_D}{\partial V_D}\right|_{\text{op}} \tag{0.430}$$

The small-signal capacitance *CD* across the diode is

$$C_D = \left.\frac{\partial Q_D}{\partial V_D}\right|_{\text{op}} \tag{0.431}$$

rD models the diode’s linear parasitic resistance.

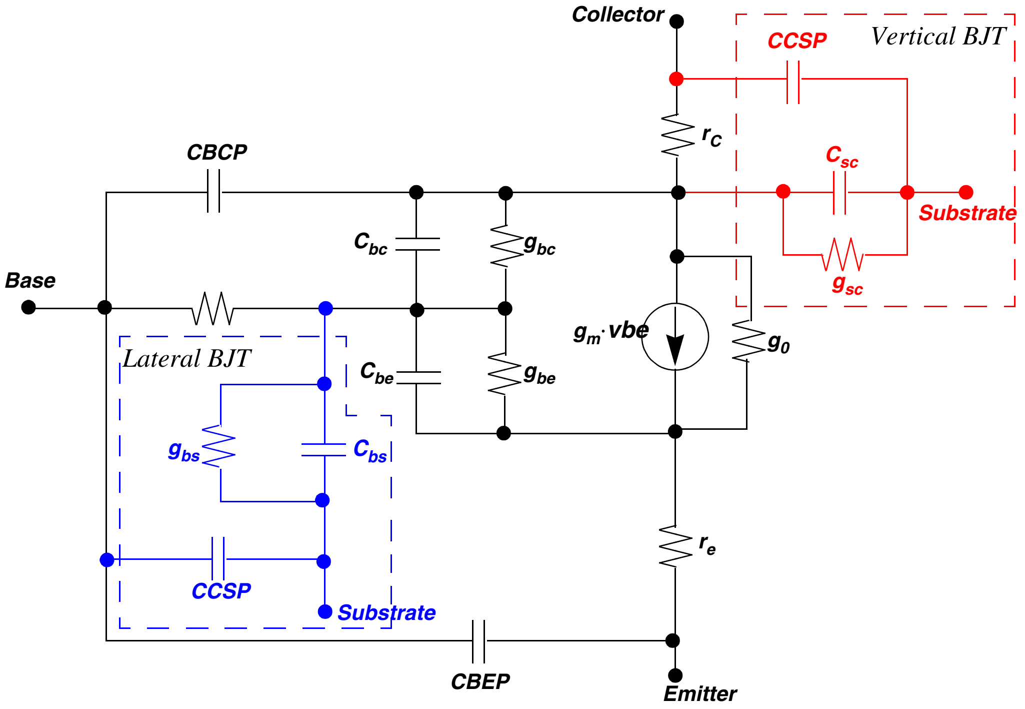
Noise Model

In addition to the thermal noise associated with *rD*, this model includes a PN junction noise source that includes shot and flicker noise:

$$\overline{\left|i_{AB_n}\right|^2} = 2qI_A \cdot \Delta f + \frac{K_f \left|I_A\right|^{A_f}}{f} \cdot \Delta f \tag{0.432}$$

BJT Level 1 (Gummel-Poon)

Small-Signal Model



The bipolar conductances are denoted by *gm*, *go*, g_π, and g_μ.

$$g_m = \left.\frac{\partial I_C}{\partial V_{BE}}\right|_{\mathrm{op}} \tag{0.433}$$

$$g_o = \frac{1}{r_o} = \left.\frac{\partial I_C}{\partial V_{CE}}\right|_{\mathrm{op}} \tag{0.434}$$

$$g_\pi = \frac{1}{r_\pi} = \left.\frac{\partial I_B}{\partial V_{BE}}\right|_{\mathrm{op}} \tag{0.435}$$

$$g_\mu = \frac{1}{r_\mu} = \left.\frac{\partial I_B}{\partial V_{BC}}\right|_{\mathrm{op}} \tag{0.436}$$

The capacitances *CBE* and *CBC* are denoted by C_π and C_μ respectively.

$$C_\pi = \left.\frac{\partial Q_B}{\partial V_{BE}}\right|_{\mathrm{op}} \tag{0.437}$$

$$C_\mu = \left.\frac{\partial Q_B}{\partial V_{BC}}\right|_{\mathrm{op}} \tag{0.438}$$

Other parameters computed are the transistor gain at DC operating point β*DC*, the AC signal gain β*AC*, and the gain-band-width product *fT*.

$$\beta_{DC} = \left.\frac{I_C}{I_B}\right|_{\mathrm{op}} \tag{0.439}$$

$$\beta_{AC} = \left.\frac{\partial I_C}{\partial I_B}\right|_{\mathrm{op}} = g_m r_\pi \tag{0.440}$$

$$f_T = \left.\frac{g_m}{2\pi(C_\pi + C_\mu)}\right|_{\mathrm{op}} \tag{0.441}$$

Gummel-Poon Noise Model

T-Spice uses the equations in this section to simulate thermal, shot, and flicker noise power densities in the BJT Gummel-Poon device model.

Thermal Noise

T-Spice uses the following equations to calculate thermal noise power density (nd_{therm}) of the base resistor (r_b), collector resistor (r_c), and emitter resistor (r_e) noise currents, respectively:

$$\begin{aligned} nd_{therm}(r_b) &= \frac{4kT}{r_{bb}} \\ nd_{therm}(r_c) &= \frac{4kT}{r_{c_{eff}}}, \\ nd_{therm}(r_e) &= \frac{4kT}{r_{e_{eff}}} \end{aligned} \tag{0.442}$$

where k is Boltzmann’s constant, T is temperature, and r_{bb}, r_{ceff}, and r_{eeff} are the effective base, collector, and emitter resistances.

Shot Noise

The following equations give the shot noise density (nd_{shot}) of the base current (I_b) and collector current (I_c), respectively:

$$\begin{aligned} nd_{shot}(I_b) &= 2qI_b \\ nd_{shot}(I_c) &= 2qI_c \end{aligned}, \tag{0.443}$$

where q is the elementary electron charge.

Flicker Noise

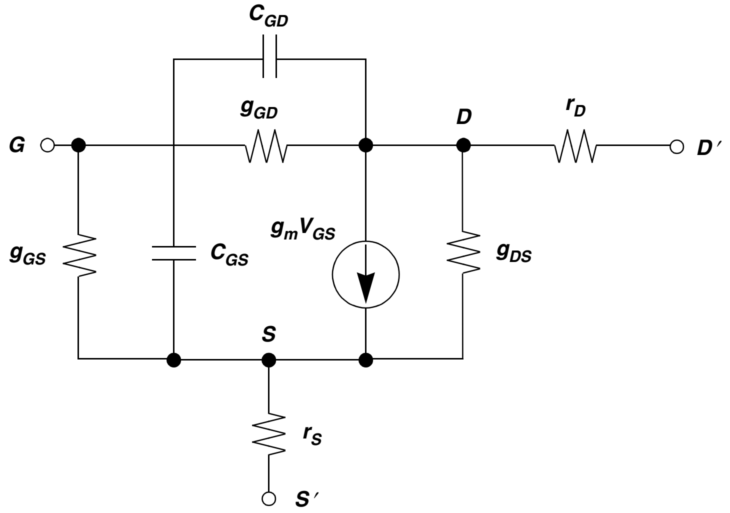
The flicker noise density (nd_{flick}) of the base current (I_b) is:

$$nd_{flick}(I_b) = \frac{KF \cdot (I_b)^{AF}}{f}, \tag{0.444}$$

where *f* represents frequency.

JFET/MESFET

Small-Signal Model



The transconductance is denoted by *gm* and the output conductance by *gDS*.

$$g_m = \left.\frac{\partial I_D}{\partial V_{GS}}\right|_{\mathrm{op}} \tag{0.445}$$

$$g_{DS} = \left.\frac{\partial I_D}{\partial V_{DS}}\right|_{\mathrm{op}} \tag{0.446}$$

The gate junction conductances are denoted by *gGS* and *gGD*. These are usually very small, since the junctions are reverse-biased in normal operation.

$$g_{GS} = \left.\frac{\partial I_{GS}}{\partial V_{GS}}\right|_{\mathrm{op}} \tag{0.447}$$

$$g_{GD} = \left.\frac{\partial I_{GD}}{\partial V_{GD}}\right|_{\mathrm{op}} \tag{0.448}$$

The gate junction capacitances are denoted by *CGS* and *CGD*.

$$C_{GS} = \left.\frac{\partial Q_G}{\partial V_{GS}}\right|_{\mathrm{op}} \tag{0.449}$$

$$C_{GD} = \left.\frac{\partial Q_G}{\partial V_{GD}}\right|_{\mathrm{op}} \tag{0.450}$$

rD and *rS* are constant linear resistances.

Noise Model

In addition to the thermal noise associated with *rD*, *rS*, and *rG*, this model includes the intrinsic noise source

$$\overline{|i_{DS_n}|^2} = \overline{|i_{ch}|^2} + \frac{K_f|I_D|^{A_f}}{f} \cdot \Delta f \tag{0.451}$$

where *ID* is the DC operating point drain current.

Channel noise is modeled in two ways, depending on the parameter **noiselevel**. When **noiselevel** < 3,

$$\overline{|i_{ch}|^2} = \frac{8}{3}kTg_m \cdot \Delta f \tag{0.452}$$

When **noiselevel** = 3,

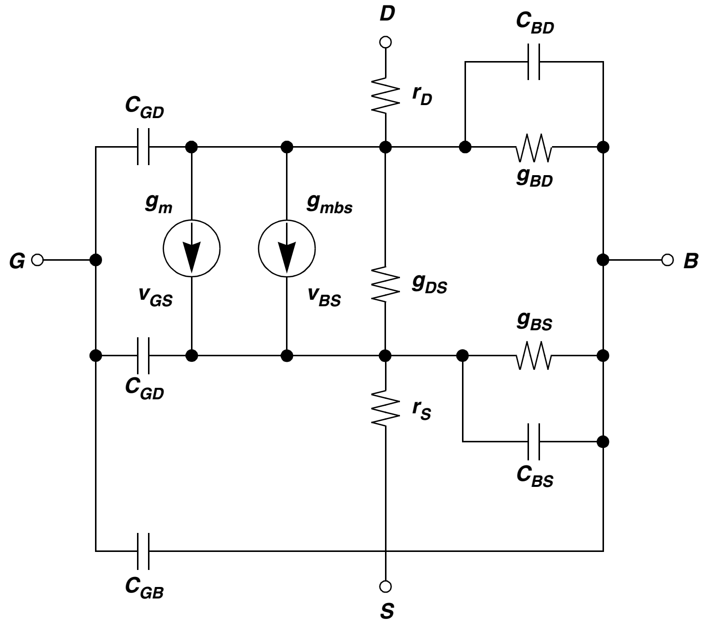
$$\overline{|i_{ch}|^2} = \frac{8}{3}kT\beta \cdot (V_{GS} - V_{t0}) \cdot G_{DS_n} \cdot \left(\frac{1+\alpha+\alpha^2}{1+\alpha}\right) \cdot \Delta f \tag{0.453}$$

where *k* is Boltzmann’s constant, *T* is the current temperature in Kelvins, *gm* is the transconductance, β is the gain at the operating point, and

$$\alpha = \begin{cases} 1 - \dfrac{V_{DS}}{V_{GS} - V_{t0}} & \textit{(linear region)} \\ 0 & \textit{(saturation region)} \end{cases} \tag{0.454}$$

MOSFET

Small-Signal Model



The MOSFET conductances are denoted by *gm*, *gDS*, and *gmbs*.

$$g_m = \left.\frac{\partial I_{DS}}{\partial V_{GS}}\right|_{\mathrm{op}} \tag{0.455}$$

$$g_{DS} = \left.\frac{\partial I_{DS}}{\partial V_{DS}}\right|_{\mathrm{op}} \tag{0.456}$$

$$g_{mbs} = \left.\frac{\partial I_{DS}}{\partial V_{BS}}\right|_{\mathrm{op}} \tag{0.457}$$

The gate junction conductances are denoted by *gBD* and *gBS*. These are usually very small, since the junctions are reverse-biased in normal operation.

$$g_{BD} = \left.\frac{\partial I_{BD}}{\partial V_{BD}}\right|_{\mathrm{op}} \tag{0.458}$$

$$g_{BS} = \left.\frac{\partial I_{BS}}{\partial V_{BS}}\right|_{\mathrm{op}} \tag{0.459}$$

The bulk junction capacitances are denoted by *CBS* and *CBD*.

$$C_{BD} = \left.\frac{\partial Q_B}{\partial V_{BD}}\right|_{\text{op}} \tag{0.460}$$

$$C_{BS} = \left.\frac{\partial Q_B}{\partial V_{BS}}\right|_{\text{op}} \tag{0.461}$$

The gate-to-junction and gate-to-bulk capacitances are denoted by *CGS*, *CGD*, and *CGB*.

$$C_{GS} = \left.\frac{\partial Q_G}{\partial V_{GS}}\right|_{\text{op}} \tag{0.462}$$

$$C_{GD} = \left.\frac{\partial Q_G}{\partial V_{GD}}\right|_{\text{op}} \tag{0.463}$$

$$C_{GB} = \left.\frac{\partial Q_G}{\partial V_{GB}}\right|_{\text{op}} \tag{0.464}$$

Since these parameters are computed directly from the MOSFET equations, the model is valid for low- and high-frequency simulations.

Noise Model

In addition to the thermal noise associated with *rD* and *rS*, this model includes *channel* and *flicker* noise sources.

$$\overline{|i_{DS_n}|^2} = \overline{|i_{ch}|^2} + \overline{|i_{flicker}|^2} \tag{0.465}$$

Channel noise is modeled in two ways, depending on the parameter **noiselevel**. When **noiselevel** < 3,

$$\overline{|i_{ch}|^2} = \frac{8}{3}kTg_m \cdot \Delta f \tag{0.466}$$

When **noiselevel** = 3,

$$\overline{|i_{ch}|^2} = \frac{8}{3}kT\beta \cdot (V_{GS} - V_{th}) \cdot G_{DS_n} \cdot \left(\frac{1 + \alpha + \alpha^2}{1 + \alpha}\right) \cdot \Delta f \tag{0.467}$$

where β is the gain at the operating point and

$$\alpha = \begin{cases} 1 - \dfrac{V_{DS}}{V_{Dsat}} & \textit{(linear region)} \\ 0 & \textit{(saturation region)} \end{cases} \tag{0.468}$$

Flicker noise is modeled in three ways, depending on **noiselevel**. When **noiselevel** = 0,

$$\overline{|i_{flicker}|^2} = \frac{K_f \cdot (i_{DS})^{A_f}}{C_{ox} L_{eff}^2 f} \cdot \Delta f \qquad (0.469)$$

When **noiselevel** = 1,

$$\overline{|i_{flicker}|^2} = \frac{K_f \cdot (i_{DS})^{A_f}}{f C_{ox} L_{eff} W_{eff}} \cdot \Delta f \qquad (0.470)$$

When **noiselevel** = 2 or **noiselevel** = 3,

$$\overline{|i_{flicker}|^2} = \frac{K_f \cdot g_m^2}{f^{A_f} C_{ox} L_{eff} W_{eff}} \cdot \Delta f \qquad (0.471)$$

References

L. W. Nagel, *SPICE2: A Computer Program to Simulate Semiconductor Circuits.* Electronics Research Laboratory Memorandum ERL-M520. Berkeley, CA: University of California, 1975.

P. Antognetti and G. Massobrio (eds.), *Semiconductor Device Modeling with SPICE*, 2nd ed. New York: McGraw-Hill, 1993.

Y. P. Tsivids, *Operation and Modeling of the MOS Transistor.* New York: McGraw-Hill, 1987.

10 User-Defined External Models

Introduction

The devices that are components of circuits simulated by T-Spice are described by *device models*. In essence, models are mathematical functions that evaluate a device's terminal currents and charges, given its terminal voltages (and in some cases other state variables).

While many "hard-coded" models are included with T-Spice, representing a wide range of devices commonly used in circuit design, the need may arise for special models not already available. Such models may represent improvements over existing models for standard devices (such as transistors), or they may describe devices not included in T-Spice (such as mechanical devices used by MEMS designers, or macro-devices to be described in a behavioral way).

The *user-defined external model* feature allows users to code custom-designed device models using C or C++. The code can be compiled into DLLs (Windows) or shared objects (Unix) and then dynamically linked to T-Spice simulations. Alternatively, the raw code can be interpreted by T-Spice *during* simulation, eliminating the compilation step with faster turnaround time for writing and debugging, at the expense of slower performance.

Creating External Models

Templates

T-Spice simulates user-defined external models with information from either of two sources:

- DLL (dynamically linked library) or shared object files, generated *before* T-Spice is run by compiling C program files. (See Compiling External Models on page 460.)
- Uncompiled C program files, interpreted by T-Spice at runtime.

The original C code must be written in any case. A basic framework for this code can be found in the file **template.c**. This file contains comment lines describing the functions and actions required of the code for a typical external model.

The heart of the code is the **ExternalModelMain()** function. This function is called whenever information about the external model is required during simulation—both for initializing and instancing a model and for evaluating devices that are instances of the model. (See Initializing the Definition on page 462 and Instancing a Device on page 462.)

ExternalModelMain() typically consists of a call to the **GetModelInfo()** function—to specify the device and/or model and the information required—and a **switch(action)** statement—to call other functions based on the value of **action** (see External Model Actions on page 468).

Examples

Five example external models are provided. Each example consists of three commented files: a C program (**.c**) file, a DLL (**.dll**) file, and a T-Spice input (**.sp**) file illustrating the use of the model (see Using External Models on page 461).

File base name	*Model description*
diode	A model incorporating most of the features of the standard SPICE diode model, with parameters analogous to those of T-Spice's internal diode model.
mos1	An implementation of the level 1 MOSFET model, with parasitic diodes and resistors, noise models, and support for table-based calculation.
resist	A simple linear resistor model.
switch	A switch model implemented as a voltage-controlled resistor with a hysteresis loop. This model is an extension of the SPICE3 voltage-controlled switch.
vco	A voltage-controlled oscillator (VCO) model.

Compiling External Models

To be used in DLL or shared object form, external models must be compiled from the original C code.

The (required) C compiler and (provided) make file, make script, and object file names for each supported platform are as follows.

Platform	*Compiler*	*Make file*	*Make script*	*Object file*
Windows	Visual C++ 4.0 or later	**model.mak**	**makemodl.bat**	**tspmodel.lib**
Solaris	gcc	**Makefile**	**makemodel_sol**	**tspmodel_sol.o**

Example files and instructions for these platforms are also provided.

To compile from a DOS command line, you must first configure your DOS environment for Microsoft Visual C++. Visual C++ includes a batch file that you can execute from autoexec.bat. This will establish the environment variables needed to run the C tools from the DOS command prompt. Generally, you can set up this environment by adding the following line to your autoexec.bat file:

```
C:\Progra~1\Micros~1\VC98\Bin\vcvars32.bat
```

To compile from the command line (Unix or DOS), type the following at the prompt:

```
script model
```

where ***script*** is the name of the appropriate make script (e.g., **mademodel.bat** on Windows) and ***model*** is the base name of the C file that contains the external model (for example, **resistor** for a model contained in **resistor.c**).

Using Microsoft Developer

To compile an external model using Microsoft Visual C++ Developer:

- With Visual C++ Developer launched, use the **File > New** command to create a new project: in the dialog, select **Projects**. Select **Win32 Dynamic Link Library**. Give the project a name and choose a **location**: use the file browser to navigate to the directory if it already exists; if not, then a new directory will be created. Click **OK**.
- When asked **What kind of DLL would you like to create?**, choose **An empty DLL project**. Click **Finish**. When the **New Project Information** dialog displays, click **OK**.
- Use the **File > New** command to create a program file: in the dialog, select **C++ Source File**. Choose a file name and click **OK**.

Note: The file name you choose must be either a C file (***.c**) or a C++ file (***.cpp**). If you choose a C++ file, you will have to wrap all T-Spice interface declarations with **extern "C"{...}**.

- Copy the text of program **template.c** and paste it into the editing space. Save the file.
- Use the **Tools > Options** command to add search paths to the T-Spice external C header and library files: in the dialog, click the **Directories** tab.
- In the **Show directories for** pull-down menu, select **Include files**. In the **Directories:** list, enter the fully qualified path to the **extmod\win32** directory. (If you used the default installation of T-Spice Pro, this will be **C:\Tanner\TSpice70\extmod\win32**.)
- In the **Show directories for** pull-down menu, select **Library files**. In the **Directories:** list, enter the same path as in the previous step.
- Use the **Project > Settings** command and perform the following actions:
- Set **Settings for** to **All Configurations**.
- Click the **Debug** tab. In the field **Executable for debug session**, enter the fully qualified path to **tspice.exe**. (**C:\Tanner\TSpice91\tspice.exe** by default.)
- In the **Working directory** field, enter the path to your T-Spice netlist.
- In the **Program arguments** field, enter the name of your T-Spice netlist.You can now compile and debug your external C model.
- Click the **C/C++** tab. From the **Category** pull-down menu, select **Code Generation**. Set processor to **Blend**. Set **Calling convention** to **__cdecl**. Set **Struct member alignment** to **8 bytes**.
- From the **Category** pull-down menu, select **Preprocessor**. Add **MAKE_DLL** to the **Preprocessor definitions** list. Set **Setting for** to both the **Win32 Debug** and **Win32 Release** versions.
- Click the **Link** tab. Make sure that **Object/library modules** includes **tspmodel.lib**.
- Click **OK**.
- Use the **Build >*project*.dll** command to compile the DLL.

Using External Models

After a user-defined external model has been created and compiled (if it is to be used in DLL or shared object form), it can be used in a simulation. This involves two procedures: initializing the model definition and instancing a device using the model.

Initializing the Definition

An external model definition is initialized from within a T-Spice input file by means of the **.model** (page 85) command with the **external** keyword:

```
.model name external ([parameter=value [...]])
```

name is the string by which the model will be identified in the input file.

Within the parentheses, predefined ***parameter***s are assigned ***value***s (as many as necessary) to tune the model's characteristics. Values may be numbers or strings; string values are enclosed in double quotes. Parameter/value pairs are separated by spaces, and are passed to the external model as *model parameters*.

One of the parameter assignments must indicate the name of the file containing the model code (in either raw or compiled form). The *name* of this parameter depends on the platform:

Platform	***Parameter***
Windows	**winfile**
Solaris	**solfile**

The *value* of this parameter is the name of the file containing the model code. T-Spice will attempt to interpret (that is, treat as raw C code) any file whose name has a **.c** extension. Any other file will be assumed to be already compiled. Compiled files (DLLs or shared objects) typically have extension **.dll** or **.sl**.

Instancing a Device

An actual *instance* of a device using an external model is created within a T-Spice input file using the **Instance (x)** (page 160) statement, in the same way that subcircuit instances are created:

```
xname node1 [node2 [...]]] modelname [parameter=value [parameter=value
    [...]]]
```

Here, ***modelname*** must match the ***modelname*** specified by the corresponding **.model** (page 85) command. Because the **x** key letter is also used for subcircuit instances, ***modelname*** should not conflict with any subcircuit definition name.

As with model initialization, parameters can be assigned values to tune the instantiated device's characteristics. Values may be numbers or strings; string values are enclosed in double quotes. Parameter/value pairs are separated by spaces, and are passed to the external model as *device parameters*.

When interpreted (C) files are used, a path must be set to the directory containing the header files. This is done in either of two ways:

- At the operating system level, set the environment variable **LUPI_INCLUDE** to the appropriate path.
- Include the following command in the input file: **.options** (page 95) **cpath=*path***, where ***path*** is the appropriate path.

External Model Features

Simple external models may only require the **EVALUATE_DEVICE** (page 473) and **EVALUATE_DERIVATIVES** (page 474) actions. More complex models may need to take advantage of advanced features, a number of which are described in this section:

- User-Defined Model and Device Parameters, below
- Error Handling on page 463
- Automatic Model Selection on page 464
- Parasitic Effects and Internal Nodes on page 464
- Tables on page 465
- Noise Analysis on page 466
- Current-Controlled Devices on page 466
- Voltage Sources on page 467

User-Defined Model and Device Parameters

When T-Spice parses a **.model** (page 85) command of type **external**, or a device statement instancing an external model, the model and device parameter names and values are stored for later interpretation.

The model’s **PARSE_MODEL_PARAMETERS** (page 469) and **PRECOMPUTE_DEVICE_PARAMETERS** (page 471) actions may use the **LookupParameter** functions to find specific parameter names and their associated values. These are then typically stored in a model-specific structure.

For example, a resistor model may contain the following type declaration:

```
typedef struct ResistorDevice
{
       double resistance;
       double mult;
} ResistorDevice;
```

The code for the **PRECOMPUTE_DEVICE_PARAMETERS** (page 471) action might be as follows:

```
ResistorDevice *r;
TspBoolean pf;
r = (ResistorDevice *)malloc(sizeof(ResistorDevice));
device->info = (void *)r;
r->resistance = LookupParameterDouble(device->parameter_list, “r|res”,
   “0.0”, &pf);
r->mult = LookupParameterDouble(device->parameter_list, “m”, “1.0”, &pf);
```

No error handling is performed in this code: if **malloc()** returns **NULL**, then disaster follows.

Error Handling

If an error occurs in the user-defined model code, T-Spice must be informed of this so that the simulation can be stopped. This is done by setting the **error** field in the **ExternalModel** or **ExternalModelDevice** structures to 1. Whenever an external model action returns to T-Spice, those

error fields are checked, and the simulation is stopped if an **error** field is nonzero. For example, the following code might be inserted in the resistor example above, immediately below the call to **malloc()**:

```
if (r==NULL)
{
        device->error = 1;
        return;
}
```

Automatic Model Selection

T-Spice allows multiple **.model** (page 85) commands in one input file with the same model name and different extensions, such as **nch.1**, **nch.2**, etc., all of which match devices with the model name **nch**. The particular **.model** command actually used with a device is determined by the *automatic model selector*.

The automatic model selector compares certain device parameter values with model parameter values to determine which model matches the device. For example, T-Spice MOSFET model parameter sets can contain parameters **wmin** and **wmax** which specify a range of transistor widths over which the parameter set is valid. The automatic model selector selects a **.model** command for each device that ensures that the device width **w** falls within the model's width range.

The **MODEL_MATCHES_DEVICE** (page 470) action implements automatic model selection. This action is executed after the **PARSE_MODEL_PARAMETERS** (page 469) action, but before any device parameter parsing. It sets the **model->model_matches_device** flag to **False** if the model cannot be matched with the device.

If no model parameter set matches a device, then T-Spice exits with an error message. If multiple model parameters match, then T-Spice selects one of the matching sets.

The following example implements automatic model selection in the case of a MOSFET:

```
MosfetModel *m;
TspBoolean pf;
double w;

m = (MosfetModel *)model->info;
w = LookupParameterDouble(device->parameter_list, "w" , "0.0", &pf);
model->model_matches_device = True;
if (m->wmax_specified && pf)
        if (w > m->wmax)
                model->model_matches_device = False;
if (m->wmin_specified && pf)
        if (w < m->wmin)
                model->model_matches_device = False;
```

Parasitic Effects and Internal Nodes

Many device models contain *parasitic* effects—effects that account for various phenomena inherent in the physical medium in which devices are used.

The **PARASITIC_SETUP** (page 472) action adds parasitic effects. Several predefined functions (**AddSeriesResistor()** (page 472), **AddParasiticDiode()** (page 472)) add simple standard parasitics. The **AddInternalNode()** (page 472) function adds custom parasitic elements by generating internal state variables. The **EVALUATE_DEVICE** (page 473) action adds the contributions of these elements.

For example, a 100 Ω parasitic resistor could be added in series with terminal 0 of a device with the **AddSeriesResistor()** (page 472) function:

```
device->rs_id = AddSeriesResistor(device, 0, 100.0, “RS”);
```

Alternatively, the **AddInternalNode()** (page 472) function could be used:

```
d->rs_terminal = AddInternalNode(device, “rs”);
```

Using **AddSeriesResistor()** (page 472) is more convenient. However, **AddInternalNode()** (page 472) allows for the addition of arbitrary series parasitic devices.

In the case of the internal node method, the resistor’s current would have to be accounted for by the **EVALUATE_DEVICE** (page 473) action as follows:

```
double irs = (device->voltage[0] - device->voltage[d->rs_terminal]) / 100.0;
device->current[0] += irs;
device->current[d->rs_terminal] -= irs;
```

In addition, the **EVALUATE_DERIVATIVES** (page 474) action would need to add the parasitic resistor’s derivative (unless numerical derivatives were used):

```
device->current_deriv[0][0] += 1.0/100.0;
device->current_deriv[0][d->rs_terminal] -= 1.0/100.0;
device->current_deriv[d->rs_terminal][0] -= 1.0/100.0;
device->current_deriv[d->rs_terminal] [d->rs_terminal] += 1.0/100.0;
```

If thermal noise modeling for the resistor were desired, then the **NOISE_SETUP** (page 474) action would set up a noise source:

```
AddNoiseSource(device, "RS", 0, d->rs_terminal, 4.0*BOLTZMANN_CONSTANT*(1.0/
    100.0)*(model->sim_data[SIM_TEMPERATURE]+KELVIN));
```

Tables

For devices with four or fewer terminals (counting internal nodes), *tables* can speed up simulation. Tables are “set up” at the beginning of simulation, but values are not filled in until needed. Thus, CPU time and memory are spent only on table entries that are actually used.

By default, external models do not use tables. To use tables, the **PRECOMPUTE_DEVICE_PARAMETERS** (page 471) action should set the **device->table_mode** flag to **True**.

Typically, a model sets the **device->table_mode** flag only if the **deftables** option **model->sim_data[SIM_DEFTABLES]** is set (nonzero). Devices may also specify device parameters which indicate whether or not tables are to be used. Devices with and without tables may be mixed in the same simulation.

Devices which share the same model may or may not be able to share the same table depending on the device parameters. For example, two MOSFETs can share a table only if they have the same length and width. If it is possible that devices may not be able to share a table, then the **TABLE_MATCHES_DEVICE** (page 471) action must be used.

The following example shows a typical **TABLE_MATCHES_DEVICE** (page 471) action:

```
MosfetDevice *d;
MosfetDevice *d_table;
```

```
d = (MosfetDevice *)device->info;
d_table = (MosfetDevice *)device->table->device->info;
if (d->width != d_table->width || d->length != d_table->length)
       model->table_matches_device = False;
```

If tables are used, then the model has control over table parameters by means of the **TABLE_SETUP** (page 472) action.

By default, the table grid will be uniformly spaced in all table dimensions, using the second terminal (terminal 1) as the reference node. The model can specify the number of grid points (default 10) and table ranges (default 5), which can be different for different table dimensions. If a non-uniform grid is to be used, the **TABLE_SETUP** (page 472) action should set up the grid.

Noise Analysis

Noise analysis is performed in conjunction with an AC analysis frequency sweep. T-Spice’s noise analysis computes the effect of “random” phenomena on the circuit output. Examples of such phenomena are resistor thermal noise and semiconductor shot and flicker noise.

Noise generators are modeled as current sources characterized by mean-square values in A^2 / Hz. Instantaneous noise source values are not available, since the effects are considered random.

The **NOISE_SETUP** (page 474) action calls the **AddNoiseSource()** (page 474) function to add a noise current source of specified mean-square value between two device terminals.

The **.print** (page 108) command with the **noise** parameter can be used (in the T-Spice input file) with the noise source name to plot the effects of such noise sources on the circuit output.

The following example adds a typical thermal noise source for a resistor:

```
AddNoiseSource(device, "RS", 0, d->rs_terminal, 4.0*BOLTZMANN_CONSTANT*(1.0/
    100.0)*(model->sim_data[SIM_TEMPERATURE]+KELVIN));
```

The following T-Spice command outputs the effect of this noise source (assuming the input file contains **.noise** (page 90) and **.ac** (page 51) commands, and a device called **x1** which uses the external model):

```
.print noise dn(x1, RS)
```

If all noise sources are frequency independent, no more code needs to be written in the external model. Otherwise, the **NOISE_EVALUATE** (page 474) action can be used to set a noise source’s mean-square value at each frequency point. This is done by calling the **SetNoiseSourceValue()** (page 475) function. The noise source is identified by the name given to it by the **AddNoiseSource()** (page 474) function.

The following example sets the flicker noise source value for a diode:

```
SetNoiseSourceValue(device, "FN", dm->kf * pow(fabs(device->current[0]),
    dm->af)/ device->model->sim_data[SIM_FREQUENCY]);
```

Current-Controlled Devices

While most T-Spice models set terminal currents and charges as functions of terminal voltages, it is sometimes desirable to use devices which use a current (instead of a voltage) as an input state variable. An example of this is the standard SPICE current-controlled current source.

T-Spice's external models allow for current-controlled devices. The **x** device statement syntax allows on its pin list either node names or voltage source currents of the form **i(*vsource*)**. For example, the declarations

```
x1 n1 n2 i(v1) mymodel
v1 n2 n3 0
```

generate a three-terminal device **x1**, with the third terminal representing the current through the voltage source **v1**.

The voltage source current can be accessed by **device->voltage[2]**.

The **device->terminal_type** array can be examined to differentiate between node voltage and voltage source current type terminals. If terminal ***i*** is a node voltage, then **device->terminal_type[*i*]** equals **TERMINAL_NODE_VOLTAGE**; if it is a voltage source current, then **device->terminal_type[*i*]** equals **TERMINAL_VSOURCE_CURRENT**.

It is illegal to set the **device->current** and **device->charge** values of a **TERMINAL_VSOURCE_CURRENT** terminal.

Voltage Sources

Voltage sources, instead of setting terminal currents and charges, specify the potential difference between two terminal nodes. The current through the device is computed internally. This requires the addition of an internal node which represents the voltage source current.

Voltage sources can be inserted between any two terminals. To set up such a voltage source, either of the **PRECOMPUTE_DEVICE_PARAMETERS** (page 471) or **PARASITIC_SETUP** (page 472) actions can call the **VoltageSource()** (page 471) function. This declares a voltage source, and internally creates the extra terminal needed to represent the voltage source current.

The following example creates a voltage source between the first two terminals:

```
d->vsid = VoltageSource(device,0,1);
```

The return value is an identification number which is later used when setting the voltage source value. The first terminal is considered the positive terminal; that is, the voltage source value is defined to be the second terminal's voltage subtracted from the first terminal's voltage.

The **EVALUATE_DEVICE** (page 473) action sets the voltage source's value by calling the **SetVoltageSourceValue()** (page 474) function.

The following example sets a constant 5 V voltage source:

```
SetVoltageSourceValue(device, d->vsid, 5);
```

A voltage source may be controlled by other terminals' voltage (or current) values. The following example generates a linear voltage-controlled voltage source (VCVS):

```
SetVoltageSourceValue(device, d->vsid, d->k*(device->voltage[2]-
    device->voltage[3]));
```

Unless numerical differentiation is used, the **EVALUATE_DERIVATIVES** (page 474) action must calculate and set a voltage source's derivatives. For independent voltage sources, that derivative is always zero, so that no model code needs to be written. But for controlled voltage sources, such as the

VCVS above, the derivatives must be set. For the VCVS example above, the derivatives would be set by:

```
SetVoltageSourceDerivative(device, d->vsid, 2, d->k);
SetVoltageSourceDerivative(device, d->vsid, 3, -d->k);
```

External Model Actions

Short descriptions of the predefined actions executed by **ExternalModelMain()** are given below. Each action is treated in more detail following. The actions are listed in the order in which they are executed. The actual *code* implementing each relevant action must, of course, be developed by the user.

PARSE_MODEL_PARAMETERS (page 469)

Read model parameter names and values and allocate memory.

MODEL_MATCHES_DEVICE (page 470)

Decide whether or not a requested model/device pair matches. Used in automatic model selection. (See Automatic Model Selection on page 464.)

PRECOMPUTE_MODEL_PARAMETERS (page 471)

Check model parameter names and values, precompute constant quantities to speed up model evaluations, and allocate memory.

PRECOMPUTE_DEVICE_PARAMETERS (page 471)

Check device parameter names and values, precompute constant quantities to speed up device evaluations, and allocate memory.

TABLE_MATCHES_DEVICE (page 471)

Decide whether or not a device may use a table which was set up for a different device.

PARASITIC_SETUP (page 472)

Add parasitics (typically series resistors, parallel diodes, or internal nodes).

TABLE_SETUP (page 472)

Compute table grid points and other parameters.

EVALUATE_DEVICE (page 473)

Given node voltages and state information, compute terminal currents and charges. *This action is required.*

EVALUATE_DERIVATIVES (page 474)

Compute all partial derivatives of the terminal currents and charges with respect to the terminal voltages. *This action is required.*

NOISE_SETUP (page 474)

Set up noise sources and make frequency-independent computations.

NOISE_EVALUATE (page 474)

Evaluate frequency-dependent noise sources at a particular frequency.

PRINT_SMALL_SIGNAL_PARAMS (page 475)

Compute and print small-signal parameter values.

CLEANUP_DEVICE (page 475)

Free memory associated with a device.

CLEANUP_MODEL (page 475)

Free memory associated with a model.

PARSE_MODEL_PARAMETERS

The **PARSE_MODEL_PARAMETERS** action processes the parameter list specified by **.model** (page 85) in the input file.

The purpose of the **.model** command is to introduce a model into the simulation. At the code level, this means initializing a data structure of type **ExternalModel**.

Most parameter names and values from the **.model** command are stored in a linked list. The pointer to this list is the **parameter_list** member of the **ExternalModel** structure.

The values from **parameter_list** must be stored as separate items of data; they are typically placed in fields of a user-defined structure whose pointer is stored by the **info** member of the **ExternalModel** structure.

Thus, the code for **PARSE_MODEL_PARAMETERS** generally involves:

- Allocating the memory needed to store specific parameter values in the particular **ExternalModel** structure being initialized. This is typically done with **malloc()**.
- Looking through (parsing) the parameter list to identify and store the specified parameter values in the appropriate locations. This is done with the **LookupParameter…()** functions.

LookupParameterDouble()

Look up parameters of type **double**. The return value is a **double**.

LookupParameterInt()

Look up parameters of type **int**. The return value is an **int**.

LookupParameterString()

Look up parameters of type **char**. The return value is a **char ***, a pointer to the string array.

The **LookupParameter…()** functions all have the same arguments:

Type	*Argument*
TspParamListElem *	**paramlist**

Type	*Argument*
	The pointer to the model's parameter list (the **parameter_list** member of the **ExternalModel** structure).
char *	**parameter_name**
	The name(s) of the parameter to be extracted from the parameter list, enclosed in double quotes. The search is case-insensitive. Multiple names for the same parameter may be searched simultaneously by separating them with a vertical bar **|** in the **parameter_name** string.
char *	**default_value**
	The value assigned to the requested parameter if it is not found in **parameter_list**, enclosed in double quotes. If necessary, this string is converted to the appropriate numeric type using **atoi()** or **atof()**. The result becomes the function's return value.
TspBoolean *	**parameter_found**
	This Boolean, passed by reference, is set to **True** if the parameter was found, and **False** otherwise.

PARSE_MODEL_PARAMETERS is executed for each **.model** command in the input file, whether the model referred to is instanced as a device or not. This is to support automatic model selection, which requires that model parameter values be known before models and devices are matched. (See Automatic Model Selection on page 464.)

PARSE_MODEL_PARAMETERS acts on model parameter sets *only*. Thus, for this action, the device (**d**) passed to **ExternalModelMain()** is **NULL**.

If **malloc()** is used to allocate the memory pointed to by **info**, then the memory should be freed by the **CLEANUP_MODEL** (page 475) action.

"Unused" model parameters—items in **parameter_list** which were never looked up—can be listed by the **PRECOMPUTE_MODEL_PARAMETERS** (page 471) action.

For models that are actually instanced as devices, any remaining initialization tasks can be performed by the **PRECOMPUTE_MODEL_PARAMETERS** (page 471) action.

MODEL_MATCHES_DEVICE

The **MODEL_MATCHES_DEVICE** action compares model and device parameter values to find a match.

- Model parameter values are looked up in **model->info** if they have been stored there (if the **PARSE_MODEL_PARAMETERS** (page 469) action has been performed already).
- Device parameter values have *not* been stored; they are found with the **LookupParameter** functions.

If the comparison fails, then the **model_matches_device** member of the **ExternalModel** structure is set to **False**.

MODEL_MATCHES_DEVICE supports automatic model selection. Several **.model** commands in one input file can specify the same model name (with different extensions), and a device can have its model chosen automatically. This is done by comparing parameters between the device and the candidate models. (See Automatic Model Selection on page 464.)

If automatic model selection is not used for a model, then **MODEL_MATCHES_DEVICE** can be omitted.

PRECOMPUTE_MODEL_PARAMETERS

The **PRECOMPUTE_MODEL_PARAMETERS** action performs preparatory computations on a model parameter set.

PRECOMPUTE_MODEL_PARAMETERS can use the following function.

PrintUnusedParameterMessage()	List “unused” model parameters—items in **parameter_list** which were never looked up.

PARSE_MODEL_PARAMETERS acts on model parameter sets *only*. Thus, for this action, the device (**d**) passed to **ExternalModelMain()** is **NULL**.

PRECOMPUTE_DEVICE_PARAMETERS

The **PRECOMPUTE_DEVICE_PARAMETERS** action parses and precomputes a device parameter set.

At this point, the device has been matched with a model, and that model is passed to **ExternalModelMain()**. (A pointer to the model is also available in the model field of the **ExternalModelDevice** structure.) Like **ExternalModel**, **ExternalModelMain()** has a **parameter_list** field and an **info** field, and their use is analogous to their **ExternalModel** counterparts.

PRECOMPUTE_DEVICE_PARAMETERS can use the following functions.

LookupParameter()	Look up device parameter values.
PrintUnusedParameterMessage()	List unused device parameters.
VoltageSource()	Set up voltage sources.

If the device is to be used in table mode, **PRECOMPUTE_DEVICE_PARAMETERS** should set **device->table_mode** to **True**. By default, **device->table_mode** is **False**.

PRECOMPUTE_DEVICE_PARAMETERS can also check **model->sim_data[SIM_DEFTABLES]** to determine the value of the **deftables** simulation option.

The maximum number of terminals for a device in table mode is four. If the device has more than four terminals, then table mode for that device is automatically turned off.

TABLE_MATCHES_DEVICE

The **TABLE_MATCHES_DEVICE** action checks whether or not a device table can be reused on another device.

Before this action, the **device->model->table** field is temporarily assigned a table. **TABLE_MATCHES_DEVICE** must decide if that table can be used with the device. The device for which the table was originally made is **device->model->table->device**.

If the **device->model->table->device** and device cannot share the table, then **TABLE_MATCHES_DEVICE** should set **model->table_matches_device** to **False**.

PARASITIC_SETUP

The **PARASITIC_SETUP** action adds parasitic devices and internal nodes.

Parasitic devices are added after all device and model parameters have been resolved. Standard MOSFET models require at least parasitic resistors and diodes.

Parasitic devices can be added in parallel (between existing nodes), or in series with a device's terminal (requiring the addition of a new node).

If the device is a MOSFET, one of T-Spice's standard MOSFET parasitic models may be used.

Care must be taken to ensure that **.print** (page 108) commands account for parasitics properly.

PARASITIC_SETUP can use the following functions.

AddSeriesResistor()

Add a resistor in series with a given terminal. The return value is an ID number which can be used later to refer to the resistor. The noise modeling for series resistors is automatic if the **noise_source_name** field is not **NULL**.

AddParasiticDiode()

Add a diode in parallel with two given terminals. The return value is an ID number which can be used later to refer to the diode. A diode model must be given as well, with a call to the **NewParasiticDiodeModel()** (page 472) function.

NewParasiticDiodeModel()

Return a required diode *model* in conjunction with **AddParasiticDiode()** (page 472), setting all parameters to typical values.

The parameter values can be modified as needed before the call to **AddParasiticDiode()** (page 472); they are the standard MOSFET parasitic diode parameters (see Additional MOSFET Parameters on page 433).

AddInternalNode()

Add a terminal to the device. The return value is the terminal number of the new terminal. The new terminal can be used to hold internal state information for the device.

For example, a series resistor can be created by **AddInternalNode()** instead of **AddSeriesResistor()** (page 472) (see Parasitic Effects and Internal Nodes on page 464). It is then the model's responsibility to compute the resistor current.

TABLE_SETUP

The **TABLE_SETUP** action sets quantities for a device table.

If the device is used in table mode (restricted to devices with 4 or fewer terminals), a table grid is computed before simulation begins. A uniform grid may be chosen, which is based simply on the number of grid points and table range (set by this action). The table grid sizes and voltage ranges are under control of the model author, and may depend on model parameter values.

Table parameters, all fields of the **device->table** structure, include:

reference_terminal

Identifies the terminal used as the potential reference (1 by default). Terminals are numbered from 0.

The table axes are the voltages at the remaining terminals with respect to the reference terminal voltage. An *n*-terminal device uses an (n–1) dimensional table.

For example, a 4-terminal device with reference terminal 1 uses (v_0–v_1, v_2–v_1, v_3–v_1) as its independent variables for the table, where v_0, v_1, v_2, and v_3 are the voltages at terminals 0, 1, 2, and 3, respectively.

grid_type

An array over the table dimensions (maximum 3) whose elements indicate what type of grid is to be used.

The options are **GRIDTYPE_UNIFORM** (uniform spacing) or **GRIDTYPE_NONUNIFORM** (non-uniform spacing, specified by the model).

number_of_points

An array over the table dimensions, each element indicating the number of table points used in the grid for a table dimension.

max_voltage

An array over the table dimensions, each element indicating the voltage range used in a uniform grid for a table dimension.

grid

An array (**double ***) over the table dimensions containing pointers to the grid points for a non-uniform grid. If a non-uniform grid is computed, then the grid point values must be sorted in increasing order.

output_type

An array over the device terminals indicating which output variables need to be saved.

The options are **OUTPUT_NORMAL** (save outputs in table), **OUTPUT_ALWAYS_ZERO** (output is always zero, no need to save in table), and **OUTPUT_COMPUTE** (output variable is computed from other output variables).

Only one terminal can be of the **OUTPUT_COMPUTE** type: this terminal's current and charge is computed by assuming that all terminals' currents and charges must sum to zero.

current_symmetry *and* **charge_symmetry**

Arrays over the table dimensions and terminals indicating which terminals' outputs have symmetry over which table dimensions. This information is used to save memory occupied by the table.

The options are **SYMMETRY_NONE**, **SYMMETRY_ODD** ($f(x)$=–$f(-x)$), and **SYMMETRY_EVEN** ($f(x)$=$f(-x)$). To save memory in a given table dimension, all outputs (both currents and charges) of a device must have symmetry (although not necessarily of the same type).

EVALUATE_DEVICE

The **EVALUATE_DEVICE** action evaluates a device's terminal currents and charges, given terminal voltage values.

The terminal voltages are stored in **device->voltage**. In the case of current-controlled sources, one of the **device->voltage** values actually represents the current through the controlling voltage source.

EVALUATE_DEVICE sets the contents of **device->current** and **device->charge**.

EVALUATE_DEVICE can use the following function.

SetVoltageSourceValue()	Set the voltage value(s), if the device to be evaluated is or contains one or more voltage sources. One of the arguments is a voltage source ID number, which is the return value of a **VoltageSource()** (page 471) function call.

EVALUATE_DERIVATIVES

The **EVALUATE_DERIVATIVES** action evaluates the partial derivatives of the device's terminal currents and charges, with respect to its terminal voltages.

EVALUATE_DERIVATIVES can use the following functions.

ComputeNumericalDerivatives()	Compute derivatives by differences. (Otherwise, if derivatives are available in analytic form, then they can be directly assigned to the matrices **device->current_deriv** and **device->charge_deriv**.)
SetVoltageSourceDerivative()	Set the derivative of a voltage value with respect to a particular device terminal voltage, if the device to be evaluated is or contains one or more voltage sources.

EVALUATE_DERIVATIVES may assume that the **EVALUATE_DEVICE** (page 473) action was performed just before with the same voltages.

If the device is in table mode, then **EVALUATE_DERIVATIVES** is never called.

NOISE_SETUP

The **NOISE_SETUP** action performs one-time initializations for noise analysis.

NOISE_SETUP can use the following function.

AddNoiseSource()	Define noise sources with assigned values.

The device's voltage, current, and charge fields contain the appropriate values at the operating point at which the noise analysis is to be performed.

At the beginning of a noise analysis, noise sources are identified for each device, and any one-time frequency-independent precomputations are performed.

NOISE_SETUP can be ignored if the model does not contain any noise source models.

NOISE_EVALUATE

The **NOISE_EVALUATE** action re-evaluates frequency-dependent noise source contributions at each frequency point of a noise analysis.

NOISE_EVALUATE can use the following function.

SetNoiseSourceValue()	Set a noise source’s mean-square value.

Any frequency-dependent noise source values are set by **NOISE_EVALUATE**; if no such noise sources are present, then **NOISE_EVALUATE** can be omitted.

PRINT_SMALL_SIGNAL_PARAMS

The **PRINT_SMALL_SIGNAL_PARAMS** action defines small-signal parameters to be printed with the **.acmodel** (page 54) command.

PRINT_SMALL_SIGNAL_PARAMS can use the **SmallSignalParameter…()** functions.

SmallSignalParameterString()	Defines a string parameter (**char***) for printing.
SmallSignalParameterDouble()	Defines a floating point (**double**) parameter for printing.
SmallSignalParameterInt()	Defines an integer parameter (**int**) for printing.

CLEANUP_DEVICE

The **CLEANUP_DEVICE** action frees any memory that has been allocated for the device structure, such as the **device->info** field.

CLEANUP_MODEL

The **CLEANUP_MODEL** action frees any memory that has been allocated for the model structure, such as the **model->info** field.

CLEANUP_MODEL is called *after* **CLEANUP_DEVICE** (page 475) has been performed for every instance of the model.

11 External Tables

Introduction

Evaluation of device models in T-Spice is based by default on “internally” generated tables, which are computed and stored during a simulation and are discarded when the simulation is over.

It is also possible to use “externally” generated tables, which are created independently, instead of within a simulation run, and can be saved for later use.

For example, tables representing commonly simulated transistors can be created once and then used in the future without having to be regenerated during each simulation run.

External table-based evaluation, while not quite as accurate as direct model evaluation, can be significantly faster; and, while not as convenient as default (internal) table-based evaluation, can provide you with more freedom and flexibility in choosing sources for table data.

For example, a device for which no model is available, but only charge and current data based on experiments, can be simulated by means of an external table incorporating that data.

The commands related to the creation and manipulation of external tables have different Windows and DOS/Unix names.

Function	*Windows*	*DOS/Unix*
Generate external tables	**Table > Generate Table** (page 513)	**tv** (page 483)
Convert tables between ASCII and binary formats	**Table > Convert Table** (page 513)	**table** (page 484)
Check table monotonicity	**Table > Monotonicity Check** (page 514)	**mc** (page 484)

Generating Tables From Presupplied Models

Where a presupplied device model is available, an external table is created with the **Table > Generate Table** command.

Generate Table takes a T-Spice input file in *table-generating* format and creates charge and current tables by (1) evaluating a model in terms of a representative device and (2) recording the results.

The table-generating input file contains information about the *model* and the *representative device*. For example, the tutorial file **p125x3.sp** looks like this:

```
* P-channel MOSFET model
.model penh pmos
+ Level=2           Ld=.03000000u       Tox=225.000e-10
+ Nsub=6.575441e+16 Vto=-0.63025        Kp=2.635440E-05
+ Gamma=0.618101    Phi=.541111         Uo=361.941
+ Uexp=8.886957e-02 Ucrit=637449        Delta=0.0
+ Vmax=63253.3      Xj=0.112799u        Lambda=0.0
+ Nfs=1.668437e+11  Neff=0.64354        Nss=3.000000E+10
+ Tpg=-1.00000      Rsh=150             Cgso=3.35e-10
+ Cgdo=3.35e-10     Cj=4.75e-04         Mj=.341
+ Cjsw=2.23e-10     Mjsw=.307
md1 n1 n2 n3 n4 penh l=1.25u w=3.0u
```

This file reproduces the *p*-channel MOSFET model parameters contained in the model file **m125cn.md**. The last line, beginning with the key letter **m**, specifies the representative MOSFET device for which the tabular data will be computed and saved.

Generate Table, using **p125x3.sp** as input, creates charge and current tables for a *p*-channel MOSFET with a length of 1.25 microns and a width of 3 microns. (The tables for a 1.25 micron-long, 2-micron-wide *n*-channel MOSFET are created in an analogous manner; the table-generating input file in the tutorial directory is **n125x2.sp**.) Devices that are of the same type but that differ in physical size (length or width) still require separate tables.

External tables are stored in either ASCII text (user-readable) or binary (machine-readable) format. The following chart compares the two formats.

Format	*Filename extensions*	*Advantages*	*Comments*
ASCII text	**.ftx** (current) **.qtx** (charge)	Portability across operating systems and platforms.	This is the default format for tables generated with the **Table > Generate Table** command.
Binary	**.f** (current) **.q** (charge)	Smaller (less storage space required); faster-loading; more precise (floating-point numbers are not truncated).	This is the default format for tables generated with the **tv** utility program.

To generate external tables from **p125x3.sp**:

Windows	*DOS/Unix*
▪ Make sure **p125x3.sp** is open, active, and saved. ▪ Select **Table > Generate Table**. ▪ The dialog suggests a table file basename, **p125x3**, which can be changed. ▪ Choose either **ASCII Text** (default) or **Binary** format. ▪ Press **Return** or click **OK**.	Type: `tv -o p125x3 -a p125x3.sp` The **-a** option generates ASCII files (the default is binary).

Tables can be converted between ASCII and binary formats with the **Table > Convert Table** command.

Generating Tables From User-Supplied Data

External tables do not have to be generated from presupplied models. Table values can be *user-supplied* from various external sources:

- Experimentally measured charge and current data. If you do not have an analytical model for the device that you want to simulate, directly measuring the charge and current values across various voltage drops and constructing tables is the only solution.
- An analytical device model you have developed or acquired which may not have an equivalent in the presupplied model library. Charge and current tables can be generated by writing a short program which solves your model equations using your parameters, and outputs charge and current values.
- The results of a T-Spice simulation on a device or circuit that you would like to define as a *macromodel* in future simulations. Transfer analysis can be used to create charge and current tables. In future simulations, the device will be seen at a high level as a “box” with certain transfer characteristics, instead of at a low level as being composed of simpler, separately modeled elements such as transistors and capacitors.

User-supplied tables must conform to the required format (see Table Format on page 479).

Using Tables

Input files using external tables differ slightly from those using internal tables. Internal tables are accessed by *model* names, while external tables are accessed by *reference* names.

For example, the MOSFET definition statements from the tutorial file **invert1.sp** are:

```
mt1 out in GND GND nenh l=1.25u w=2u
mt2 out in Vdd Vdd penh l=1.25u w=3u
```

These statements specify model names, **nenh** and **penh**, to be used in the generation of *internal* tables. *External* tables, which already exist when the simulation begins, are specified by reference names — for example, **nmost** and **pmost** (any legal names could be used). If **invert1.sp** were modified to use external tables, the corresponding lines would be:

```
mt1 out in GND GND nmost l=1.25u w=2u
mt2 out in Vdd Vdd pmost l=1.25u w=3u
```

Of course, T-Spice needs to know *where* to find the appropriate tabulated data when a particular reference name is given. This information is supplied by the **.table** command. The following lines would be added to the modified **invert1.sp**:

```
.table nmost n125x2.ftx n125x2.qtx
.table pmost p125x3.ftx p125x3.qtx
```

The **.table** command associates the reference names (**nmost** and **pmost**, used by the MOSFET device statements) with existing table files (**n125x2.qtx**, **n125x2.ftx**, **p125x3.qtx**, and **p125x3.ftx**). The *reference* names need not resemble the *file* names. When the associations have been made, T-Spice searches the input file for device statements which call the same reference names, and so pairs specific devices with appropriate charge and current tables.

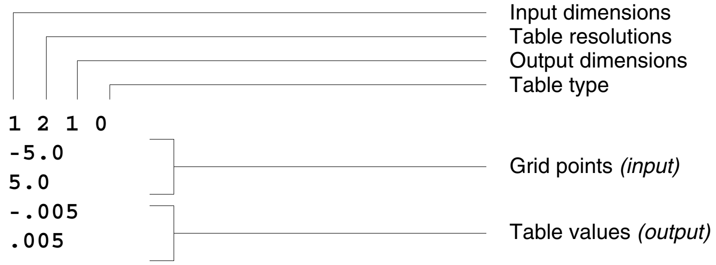
Table Format

A T-Spice external device table lists either charge or current values which vary according to the different voltages between terminals on the device.

Tables have one or more *input dimensions*. Each input dimension consists of a set of voltages at one of the device terminals with respect to the *reference terminal*. (In the case of tables used in macromodeling, the reference terminal is the *second* terminal named on the ***Xinstance*** command.)

For example, the tables for a device with three terminals **a**, **b**, and **c**, with **a** as the reference terminal, would have two input dimensions, corresponding to V(**b**,**a**) and V(**c**,**a**). Thus, one column in the charge table would list charges as a function of the voltage between **b** and **a**; the other column, between **c** and **a** (and similarly for the current table).

Tables contain other information as well. The example below — the steady-state current table for a linear resistor — is followed by a detailed description of the format and order of entries in a table. Entries in the table are separated by spaces, tabs, or new lines.



```
1 2 1 0
-5.0
5.0
-.005
.005
```

Input dimensions [integer]. This must be the first entry. The number of input dimensions is the number of pairwise combinations of terminals with the reference terminal, or (*n*–1) dimensions for an *n*-terminal device. (The reference terminal itself is not counted, since the corresponding voltage is always zero.)

Table resolutions [integer]. One resolution value is required for each dimension; it is the number of voltage (input) values, or *grid points*, the table will list in that dimension. (These entries are equivalent in function to the **.gridsize** command in input files.)

Output dimensions [integer]. The number of output dimensions is the number of distinct charge or current (output) values needed to characterize the device's behavior. This is usually one fewer than the number of terminals on the device, due to charge/current conservation: once (*n*–1) currents or charges are known for an *n*-terminal device, the final terminal's values can be readily computed.

Table type [integer]. The table type indicates the device type, as follows:

Table type	***Device type***	***Description***
0	MACRO	A device whose table is not generated from a predefined model but by some other means, such as macromodeling.
1	NMOS	*N*-channel MOSFET
2	PMOS	*P*-channel MOSFET
3	D	Diode
4	NPN	npn
5	PNP	pnp
6	NJF	*N*-channel JFET
7	PJF	*P*-channel JFET
10	NMF	*N*-channel MESFET
11	PMF	*P*-channel MESFET

Grid points [float]. The list of voltage (input) values used to determine the charge and current (output) values. The full set of grid points for the first input dimension is listed first, then the next, and so on. The total number of grid points is the sum of the table resolution values.

Table values [float]. The list of charge and current (output) values corresponding to the voltage (input) values. The table values are sorted into vertical columns, one column per output dimension. The order of the columns is specified in the input file by the device statement: the first terminal listed there uses the first (leftmost) column, the second terminal uses the second column, and so on. (The last terminal's charges and currents are computed, not tabulated.) Each column lists currents or charges as if the voltages were being *swept* through their grid ranges (see Multiple Input Dimensions, below). The total number of table values *per column* is the product of the table resolutions.

Grid Point Resolution

Grid points need not be evenly spaced. A higher density of grid points can be taken around areas where device behavior is non-linear, for higher accuracy.

Table Value Monotonicity

For DC and transfer analyses, steady-state currents must be monotonic as a function of voltage. For transient analyses, charges must be monotonic in voltage as well.

The **Table > Monotonicity Check** command helps to locate areas of non-monotonicity. **Monotonicity Check** determines whether or not the table values for each combination of indices in each dimension are monotonic. Any non-monotonicities found are counted, and the table coordinates at which they were found are reported (*Windows*: to a specified output file; *DOS/Unix*: to standard output unless redirected).

Multiple Input Dimensions

In the case of more than one input dimension, the table values are listed as follows: Each output column lists table values as if the voltages were being *swept* through their ranges, so that a table value is given for *each* combination of voltages. For example, suppose that a device's table has *three* input dimensions (**V1**, **V2**, **V3**), each with *two* grid points (denoted by subscripts **a** and **b**). Then the currents in *each* column, computed from combinations of the three voltages, are listed in the following order:

```
I(V1a,V2a,V3a)
I(V1b,V2a,V3a)
I(V1a,V2b,V3a)
I(V1b,V2b,V3a)
I(V1a,V2a,V3b)
I(V1b,V2a,V3b)
I(V1a,V2b,V3b)
I(V1b,V2b,V3b)
```

MOSFET Tables

The format for MOSFET tables is slightly different to reduce storage and access time.

- Only the drain and source steady-state currents are stored as table (output) values.
- Charge and current values are listed only for certain ranges of *Vds* (voltage between drain and source). The drain and source terminals are defined arbitrarily, so it is redundant to store both positive and negative *Vds* grid points.
- Only positive values for NMOS and negative values for PMOS are stored. The table type indicates whether the table is for a NMOS or a PMOS transistor.

Otherwise the format is standard. For example, NMOS charge tables take the following form, where ***A***, ***B***, ***C*** denote the numbers of grid points in the three input dimensions (*Vds*, *Vgs*, *Vbs*, respectively), ***V*** denotes voltage, and ***Q*** denotes charge:

```
3      A      B      C
3      1

Vds,1  Vds,2  . . .  Vds,A
Vgs,1  Vgs,2  . . .  Vgs,B
Vbs,1  Vbs,2  . . .  Vbs,C

Qd(Vds,1,Vgs,1,Vbs,1)Qs(Vds,1,Vgs,1,Vbs,1)Qg(Vds,1,Vgs,1,Vbs,1)
Qd(Vds,2,Vgs,1,Vbs,1)Qs(Vds,2,Vgs,1,Vbs,1)Qg(Vds,2,Vgs,1,Vbs,1)
. . .
Qd(Vds,A,Vgs,B,Vbs,C)Qs(Vds,A,Vgs,B,Vbs,C)Qg(Vds,A,Vgs,B,Vbs,C)
```

The steady-state current file is similarly structured, except that the values for the gate are excluded and the number of output dimensions is two instead of three. The default input dimensions are: ***A*** (*Vds*) = 9, ***B*** (*Vgs*) = 15, and ***C*** (*Vbs*) = 4. These dimensions can be modified with the **.gridsize** command. The default voltage range is [0.0,+5.0] for NMOS and [–5.0,0.0] for PMOS. These ranges can also be modified with the **.vrange** command. Examples of PMOS and NMOS charge and current tables (**p125x3.ftx**, **p125x3.qtx**, **n125x2.ftx**, **n125x2.qtx**) are included.

The charge and current tables for a transistor should have the same grid points (to lower lookup times); if this is not the case, then the transistor will be treated as a general four-terminal device. T-Spice will automatically reduce the dimensionality of a transistor table if the drain or source is connected to the bulk (to lower lookup and interpolation times).

The ordering of the nodes on a table ***Xinstance*** command corresponds to the ordering of the columns in the table. If a node has only zero values for all regions of operation — for example, if the current at a gate is zero — then the **.ftx** table should be modified as follows: (1) list the gate last; (2) reduce the number of output dimensions by one; (3) omit the table values for the gate.

A Simple Example

The following example is the steady-state current table, **res.ftx**, for a linear resistor (10 kilohms).

```
1 2 1 0
-5.0 5.0
-.005
.005
```

The first line of the table contains the first four required entries.

- The number of *input dimensions* is **1**, since a resistor has two terminals, and one is used as the reference. (In other words, the only voltage of interest is the one across the resistor’s two terminals.)
- One *table resolution* value, corresponding to the single input dimension, is given: the table’s first (and only) dimension contains **2** grid points.
- The number of *output dimensions* is **1**, since currents need to be specified at only one terminal of the resistor (the current at the other terminal can be computed from the first).
- The *table type*, **0**, indicates a device which is not represented in the T-Spice model library.

The *grid points*, **–5.0** and **5.0** volts, establish fixed points from which the current at any other voltage can be computed by linear interpolation and extrapolation. The full set of grid points in the first input dimension is listed first, then the second (if there is a second input dimension), and so on.

Following the grid points is a column of *table values*. Two values are given, corresponding to the two input voltages in the first (and only) dimension, and representing the current at one of the terminals given those voltages.

The resistor's charge table is stored in another file, **res.qtx**. The resistor does not have charge associated with it, but a charge table is still required, one that contains zero values:

```
1 2 1 0
-5.0 5.0
0.0
0.0
```

Since the standard resistor device statement does not include an option for specifying an alternate model, this resistor would be implemented as a *macromodel* (see below) by including the **.table** command in the circuit description using the tabulated resistor. Then, an instance of a table device is created using the ***Xname*** subcircuit-device command.

```
.table myres res.ftx res.qtx
xr1 in out myres
```

where **xr1** is the name of the resistor in the circuit, **myres** is the table reference name, and **in** and **out** are the resistor's two terminals.

Utility Programs

Several utility programs are included with T-Spice; these aid in the creation and manipulation of external tables. Giving the name of any of the utility programs (**tv**, **table**, **mc**) *without* arguments at the command prompt produces a usage message.

tv

tv, a *table generator*, generates external charge and current tables from an input file consisting of a **.model** command with parameters and a statement specifying a representative device.

```
tv [-aEUV] -o output input
```

Option	*When on*	*When off (default)*
-a	Generates tables in ASCII text format.	Generates tables in binary format.
-E	Evaluation mode. Prompts for a set of (input) terminal voltages, and returns the (output) terminal charges and currents at those voltages.	—
-U	Prints a usage message and quits the program.	—

Option	*When on*	*When off (default)*
-V	Prints the version number and acknowledgments and quits the program.	—
-o *output*	Prints results to table files with base name ***output***.	[This argument is required.]
input	Uses file ***input*** to generate tables. Input files typically have the extension **.sp**.	[This argument is required.]

table

table, a *table format converter*, converts external charge and current tables between ASCII text (user-readable) and binary (machine-readable) formats.

```
table [-UV] input output
```

Option	*When on*	*When off (default)*
-U	Prints a usage message and quits the program.	—
-V	Prints the version number and acknowledgments and quits the program.	—
input	Looks in file ***input*** for the table to be converted. ASCII table files must have extension **.ftx** or **.qtx**; binary table files must have extension **.f** or **.q**.	[This argument is required.]
output	Writes the converted table to file ***output***. ASCII table files must have extension **.ftx** or **.qtx**; binary table files must have extension **.f** or **.q**.	[This argument is required.]

mc

mc, a *monotonicity checker*, determines for a table whether or not the table values for each combination of indices in each input dimension are monotonic. Any non-monotonicities found are counted, and the table coordinates at which they were found are reported to standard output (unless redirected).

```
mc [-cpUV] input
```

Option	*When on*	*When off (default)*
-c	Continues checking (even if non-monotonicity is found).	Stops checking as soon as non-monotonicity is found.
-p	Prints out grid values along each dimension instead of checking for monotonicity.	—

Option	*When on*	*When off (default)*
-U	Prints a usage message and quits the program.	—
-V	Prints the version number and acknowledgments and quits the program.	—
input	Looks in file ***input*** for the table to be checked. ASCII table files must have extension **.ftx** or **.qtx**; binary table files must have extension **.f** or **.q**.	[This argument is required.]

12 Parametric Analysis

Introduction

T-Spice is often required to study the effects of *variations* in parameter values on circuit performance. For example, parametric analysis can be used to evaluate multidimensional trends in the output over defined ranges of input values, or the sensitivity of circuit behavior to random fluctuations in fabrication conditions.

A large range of parameters may be systematically and automatically varied:

- External parameters (such as temperature)
- Simulation parameters (such as tolerances)
- Device parameters (such as input voltage level or transistor length)
- Model parameters (such as transistor threshold voltage)

Three types of parametric analysis are supported by T-Spice: *parameter sweeping*, *Monte Carlo analysis*, and *optimization*.

This chapter will guide the T-Spice Pro user through several tutorial problems in order to demonstrate some basic concepts of parametric analysis.

The example files for these tutorials can be found in the **tspice91\examples\input** subdirectory of the T-Spice Pro installation path.

Output File Formats

Results produced by parameter sweeps and Monte Carlo analysis are easily read in T-Spice output files. You can open a T-Spice output file from the T-Spice Simulation Manager by clicking on the **Show Output** button. Alternatively, select **File > Open** and browse within the **Open** dialog to select the output filename. T-Spice output files are text files; they are also readable in the text editor of your choice.

Analysis results for parameter sweeps are reported in table format, with section headings that contain the current values of the swept parameters. For example, if a **.step** command invokes several transient analyses, each analysis produces its own output section, the header of which shows the parameter values for that analysis (e.g., **TRANSIENT ANALYSIS – vdd=3**).

Results obtained from **.measure** commands are listed at the end of each output section corresponding to a specific parameter value. These measurements are summarized at the end of the sweep in a table labeled **TRANSFER ANALYSIS**. The use of a **.step** command causes **.measure** results to be plotted in W-Edit, with the swept variable as the *x*-axis.

Parameter Sweeps

In a *parameter sweep*, a specified parameter is held or initialized at a given value, all analyses requested by the input file are performed, and the results are recorded. Then the parameter is incremented by a set amount, and the same analyses are repeated. The cycle continues as the parameter is incremented through a defined range of values.

Parameter values may be swept *linearly*—in identical increments, typically through a limited range—or *logarithmically*—in exponential increments, typically through a range spanning multiple orders of magnitude. You can also specify a sweep over a list of values.

Parameter sweeping is performed by using the **sweep** option with one of the following commands:

- **.ac** (see .ac on page 51)
- **.dc** (see .dc on page 60)
- **.step** (see .step on page 128)
- **.tran** (see .tran on page 137)

Additionally, it is also possible to perform simultaneous parameter sweeps of several different variables using the command **.data** (see .connect on page 57).

Adding the **sweep** option to a **.tran** or **.ac** command instructs T-Spice to perform that analysis for all parameter values of the specified sweep. Using the **sweep** option with an analysis command is similar to using it with **.step**. A single **.step** command causes T-Spice to perform parameter sweeps for *all* analysis commands in the input file. If **sweep** is specified on an analysis command and a **.step** command is present, the **sweep** specified with the analysis command is nested inside the sweep specified with the **.step** command.

All input files listed for this chapter are in the directory **<install_dir>\tutorial\input**. If you chose default installation, this is equivalent to **C:\Tanner\TSpice91\tutorial\input.**

Example 1: Parametric Sweep

This example uses a ring oscillator to demonstrate the basic features of a parametric sweep.

T-Spice Input	**ring2.cir**
Output	**ring2.out**

T-Spice Input

```
* Circuit: ring2.sp
*
.SUBCKT inv in out Gnd Vdd
c2 out Gnd cap
m1p out in Vdd Vdd pmos L=5u W=12u
mn1 out in Gnd Gnd nmos L=5u W=8u
.ENDS

* Main circuit: ring2
cinv1 a7 Gnd 400ff
.include ml2_125.md
```

```
Xinv1 a1 a2 Gnd Vdd inv
Xinv2 a2 a3 Gnd Vdd inv
Xinv3 a3 a4 Gnd Vdd inv
Xinv4 a4 a5 Gnd Vdd inv
Xinv5 a5 a6 Gnd Vdd inv
Xinv6 a6 a7 Gnd Vdd inv
Xinv7 a7 a1 Gnd Vdd inv
.measure tran period trig v(a2) val=3.0 fall=2 targ v(a2) val=3.0 fall=3
.measure tran pulsewidth trig v(a2) val=1.5 rise=2 targ v(a2) val=1.5 fall=2
.measure tran timedelay trig v(a2) val=3.0 fall=2 targ v(a1) val=3.0 fall=2
.param cap=800ff
.print tran a1
.step cap 200f 1000f 200f
.tran/powerup 1n 800n
vdd Vdd Gnd 3.0
* End of main circuit: ring2
```

In this example, instead of keeping the load capacitor in the inverter subcircuit constant, the capacitor is defined as a variable **cap** and T-Spice sweeps it over a range of **200fF** to **1000fF**. The **.param** statement sets a nominal value for **cap** and the **.step** command sweeps **cap** linearly from **200fF** to **1000fF** in increments of **200fF**.

The **.measure** statement measures the period, time delay, and pulse width of the ring oscillator at the different values of capacitance. In this example, period is measured at **v(a2)=3V** from its second falling edge to its third falling edge. Time delay is measured from the second falling edge of **v(a2)** at **3V** to the second falling edge of **v(a1)** at **3V**. Pulse width is measured from the second rising edge of **v(a2)** at **1.5V** to the second falling edge of **v(a2)** at **1.5V**.

Output

T-Spice reports the transient analysis results in five sections for **cap=200fF**, **400fF**, **600fF**, **800fF**, and **1000fF**.

Following is part of the output section for **cap=200fF**.

```
TRANSIENT ANALYSIS - cap=2e-013
Time<s>           v(a1)<V>
0.0000e+000  0.0000e+000
1.0876e-010  1.9456e-002
4.9424e-010  3.3521e-001
8.9842e-010  6.0502e-001
1.2047e-009  7.0183e-001
```

The measurement results are reported at the end of each section.

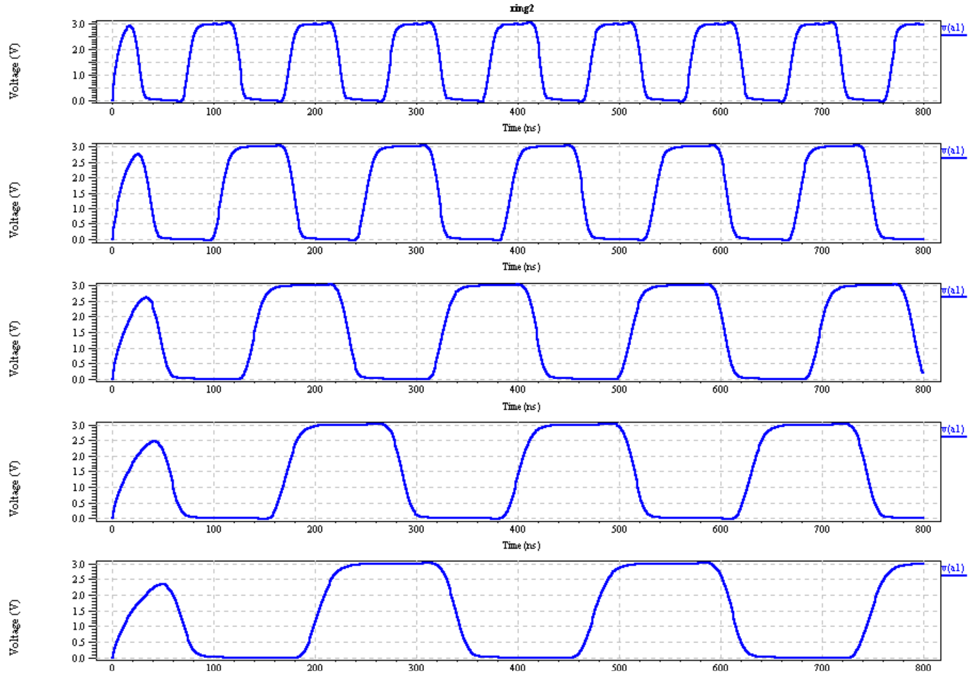
```
MEASUREMENT RESULTS - cap=2e-013

period = 9.9026e-008
   Trigger = 1.7549e-007
   Target  = 2.7452e-007

pulsewidth = 4.8288e-008
   Trigger = 1.3217e-007
   Target  = 1.8046e-007

timedelay = 4.1316e-008
   Trigger = 1.7549e-007
   Target  = 2.1681e-007
```

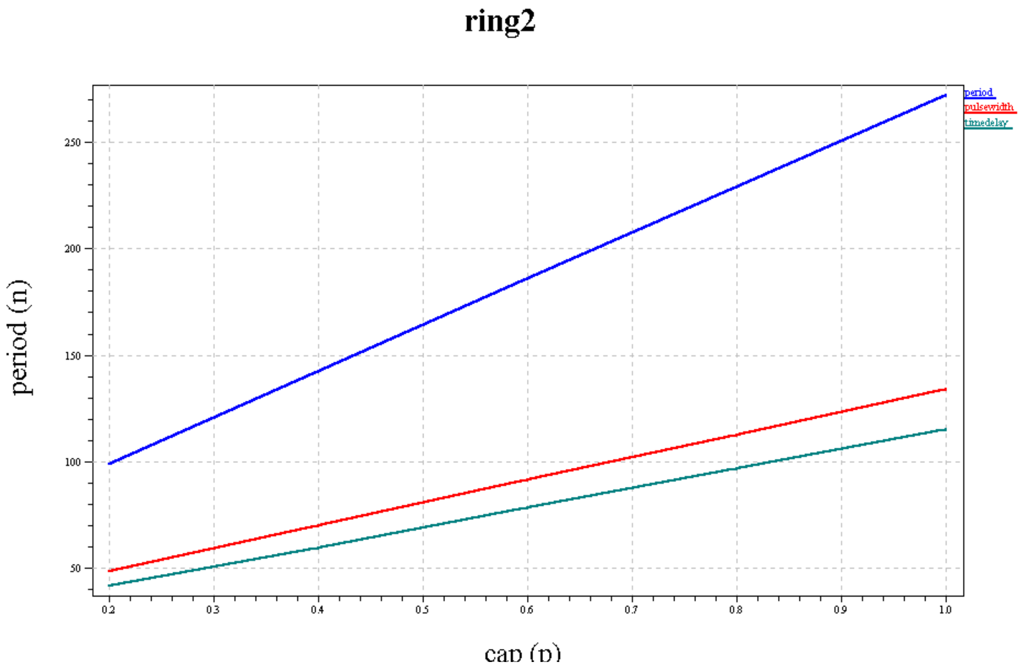
Measurement results are summarized in a table at the end of the output file. Because the **.step** command was used, measurement results will be plotted against the parameter **cap** in W-Edit.

```
TRANSFER ANALYSIS
       cap<>     period<>  pulsewidth<>  timedelay<>
 2.0000e-013  9.9026e-008  4.8288e-008  4.1316e-008
 4.0000e-013  1.4248e-007  6.9783e-008  5.9662e-008
 6.0000e-013  1.8579e-007  9.1213e-008  7.8093e-008
 8.0000e-013  2.2913e-007  1.1262e-007  9.6608e-008
 1.0000e-012  2.7235e-007  1.3401e-007  1.1505e-007
```

Waveform

The following two figures show the waveform output. The first displays the output of the ring oscillator vs. time with different capacitance values; the second displays the period, time delay and pulse width in a single chart.


ring2
Voltage (V)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500
600
700
800
Time (ns)
v(a1)


ring2
period (n)
250
200
150
100
50
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
cap (p)
period
pulsewidth
timedelay

Monte Carlo Analysis

In a Monte Carlo analysis, T-Spice performs simulation runs using randomly chosen parameter values. The parameter values for each run are chosen from probability distributions defined by the user.

T-Spice's Monte Carlo analysis will generate a summary output of all simulation run measurement results after all runs are completed. T-Spice will also report, for each Monte Carlo iteration, the values of all expressions evaluated using probability distributions.

Monte Carlo analysis can be invoked using the keywords **sweep monte** with one of the following commands:

- **.ac** (page 51)
- **.dc** (page 60)
- **.step** (page 128)
- **.tran** (page 137)

Syntax and options for the keyword **sweep** are defined in the section describing **.step** (page 128).

Probability distributions are assigned to parameters by use of the **.param** command. For a complete description of the syntax of this command, see **.param** (page 102).

Example 2: Monte Carlo Analysis

This example demonstrates Monte Carlo Analysis on a CMOS inverter circuit.

Input	**invert5.cir**
Output	**invert5.out**

Input

```
...
* Main circuit: invert5
c2 out Gnd 800ff
.include ml2_125mc.md
m1n out in Gnd Gnd nmos L=5u W=8u
m1p out in Vdd Vdd pmos L=5u W=12u
.measure tran falltime trig v(out) val=2.8 fall=1 targ v(out) val=0.2 fall=1
.param vto_n=unif(0.622490, 0.5, 1) vto_p=unif(-0.63025, 0.5, 1)
.tran 2n 600n sweep monte=10
.print tran in out
vdd Vdd Gnd 3.0
vin in Gnd pwl (0ns 0V 100ns 0V 105ns 3V 200ns 3V 205ns 0V 300ns
+ 0V 305ns 3V 400ns 3V 405ns 0V 500ns 0V 505ns 3V 600ns 3V)
* End of main circuit: invert5
```

A Monte Carlo analysis sweeps parameter values that are chosen based on statistical variations. In this example, T-Spice varies the model parameter **vto** using random values chosen by probability distribution. In model file **ml2_125mc.md**, the **.model** statement specifies the **vto** parameter as two variables: **vto_n** for an n-channel MOSFET and **vto_p** for a p-channel MOSFET.

The **.param** statement defines the probability distribution, where **vto_n=unif(0.622490, 0.5, 1)** and **vto_p=unif(-0.63025, 0.5, 1)** select uniform distributions centered at **0.622490** and **-0.63025** with relative variation of **50%**. The keyword **monte=10** in the **.tran** statement invokes Monte Carlo analysis with 10 runs. The **.measure** statement measures the falltime of the output pulse for different values of **vto**.

Output

T-Spice reports the transient analysis results in ten sections for the ten Monte Carlo runs.

Following is part of the output section for the first run.

```
TRANSIENT ANALYSIS - Monte-Carlo-index=1
Time<s>            v(in)<V>    v(out)<V>
   0.0000e+000  0.0000e+000  2.9996e+000
   6.0000e-010  0.0000e+000  2.9996e+000
   2.6000e-009  0.0000e+000  2.9996e+000
   4.6000e-009  0.0000e+000  2.9996e+000
   6.6000e-009  0.0000e+000  2.9996e+000
   8.5999e-009  0.0000e+000  2.9996e+000
...
```

Measurement results are reported at the end of each section.

```
MEASUREMENT RESULTS - Monte-Carlo-index=1

falltime = 1.2278e-008
   Trigger = 1.0471e-007
   Target  = 1.1699e-007
```

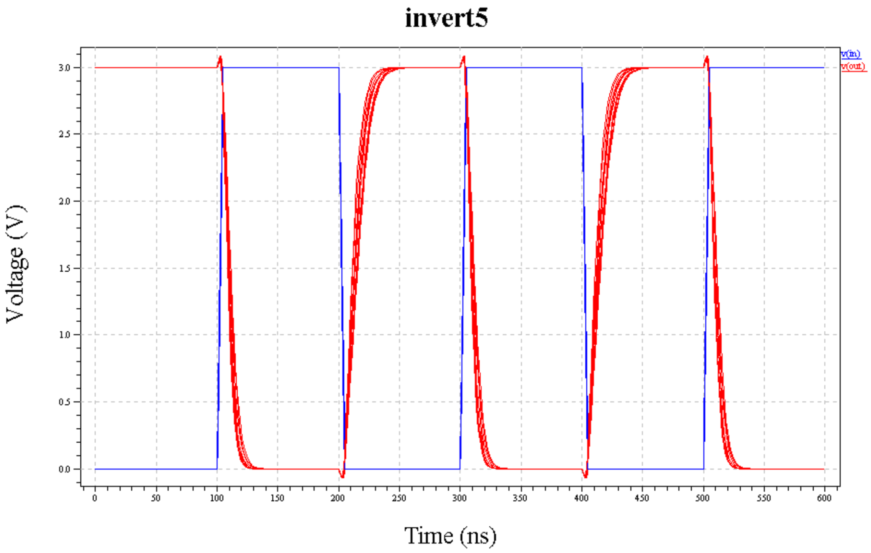
At the end of the output file, T-Spice reports the Monte Carlo values for each run The measurement results are summarized along with statistical results from the analysis (minimum, maximum, mean, average deviation, variance, and sigma).

```
MONTE CARLO PARAMETER VALUES

Index 1
              parameter Vto for model nmos =   3.1202e-001
              parameter Vto for model pmos =  -6.7032e-001

Index 2
              parameter Vto for model nmos =   4.3157e-001
              parameter Vto for model pmos =  -8.2483e-001

Index 3
              parameter Vto for model nmos =   6.7541e-001
              parameter Vto for model pmos =  -6.1756e-001
...


TRANSFER ANALYSIS
      Index<>   falltime<>
  1.0000e+000  1.1738e-008
  2.0000e+000  1.2477e-008
  3.0000e+000  1.4346e-008
  4.0000e+000  1.3173e-008
  5.0000e+000  1.5912e-008
  6.0000e+000  1.2399e-008
  7.0000e+000  1.5150e-008
```

```
8.0000e+000  1.2944e-008
9.0000e+000  1.2069e-008
1.0000e+001  1.2278e-008
    Minimum  1.1738e-008
    Maximum  1.5912e-008
       Mean  1.3249e-008
     Avgdev  1.1325e-009
   Variance  1.9941e-018
      Sigma  1.4121e-009
```

Waveform

The following figure displays families of output traces from a Monte Carlo analysis.



Optimization

The T-Spice optimization feature allows circuit parameters to be tuned within given ranges to achieve the best possible circuit performance. T-Spice seeks to minimize the difference between performance measurements and a user-defined optimization goal. In order to specify an optimization, you must supply the following:

[1] A list of parameters which can be adjusted to optimize performance. Each parameter is assigned a nominal, minimum, and maximum value with **.paramlimits**. The nominal value serves as the initial guess in the optimization process. (For a complete description of the syntax of the **.paramlimits** command, see **.paramlimits** (page 106))

[2] An optimization goal defined in terms of **.optgoal** commands and **.measure** results. T-Spice attempts to minimize the deviation of **.measure** results from their respective goal values defined in **.optgoal**. If multiple **.optgoal** commands are used, each **.optgoal** parameter has a weight value to indicate its

relative importance. (For a complete description of the syntaxes, see **.macro /.eom** (page 77) or **.optgoal** (page 92).)

[3] A **.optimize** command that invokes the optimization run. The optimization model and analysis name are specified using the **.optimize** command. (For a complete description of the **.optimize** command syntax, see **.optimize** (page 93).)

[4] A **.model** command which defines the optimization algorithm, as well as optimization algorithm parameters and tolerances. (For a complete description of the **.model** command syntax, see **.model** (page 85).)

Defining Optimization Parameters

Any parameter defined using **.param** can be used as an optimization parameter. Use the **.paramlimits** command to associate such a parameter with an optimization run, so that T-Spice can vary it during the optimization. There is no limit on how many optimization parameters an optimization run can have, but the optimization will be much faster with fewer parameters. The precise syntax for this keyword is described in **.paramlimits** (page 106).

Defining Optimization Goals

Optimization goals are defined using **.optgoal**. The function of the **.optgoal** command is to link a **.measure** result to an optimization run and to specify a goal value for the measurement. During each optimization run, T-Spice calculates an optimization function based on the differences between measurements and their corresponding optimization goals. T-Spice uses this calculation to find parameter values that minimize the deviation of measurement results from goal values.

An optimization run can have multiple **.optgoal** commands, each of which is assigned a weight. The value of the optimization function for each run is obtained by summing the weighted individual goals: each **.optgoal** contributes a term of the form *weight* × *(goal-result)/goal*.

Curve-fit optimization can be performed by using the error function measurements available on the **.measure** command.

For more information on the relevant commands, see **.macro /.eom** (page 77) and **.optgoal** (page 92).

Invoking Optimization

A **.optimize** command invokes an optimization run using the parameters and goals specified by **.paramlimits** and **.optgoal** when these commands use a matching optimization run name. Only those parameters for which the run name matches are varied during the optimization run. The **model** keyword defines an optimization model, which is specified using a **.model** command with optimization algorithm parameters such as iteration count limits and convergence tolerances. The ***analysisname*** option identifies the **.step**, **.ac**, **.dc**, or **.tran** analysis that will be performed to evaluate the measurements for the optimization.

After an optimization, the optimized parameter values are used in all subsequent analyses specified in the same input file. This allows for *incremental optimization*: some parameters can be optimized while others are held fixed; other parameters can then be optimized based on the results of the first optimization. DC analyses are performed first, followed by AC analyses, and then transient analyses. Multiple analyses of the same type are performed in the order in which they are found in the input file.

The T-Spice wizard-style user interface will guide you in setting up optimization commands.

Example 3: Optimization

Input A	**opamp_ac.cir**
Input B	myopamp.cir
Output	**myopamp.out**

This tutorial will demonstrate the procedures to be followed in setting up an optimization command. We use the operational amplifier introduced in Example 4 to show how you can optimize output by varying the output transistor’s lengths and widths. We can use the existing netlist by adding optimization commands to the netlist.

This exercise uses the input file **opamp_ac.cir** as a base file to which we will add optimization commands in the course of the tutorial. The modified input file that you create will be named **myopamp.cir**. For reference, we have included a file called **opamp4.cir**, which represents what **myopamp.cir** should look like at the end of the tutorial.

☑ Open **opamp.cir** and choose **File > Save As** to rename the file. Name the file **myopamp.cir**.

☑ For simplicity, remove the statements **.print ac vp(out)** and **.acmodel opamp1m.out (*)** in the original netlist.

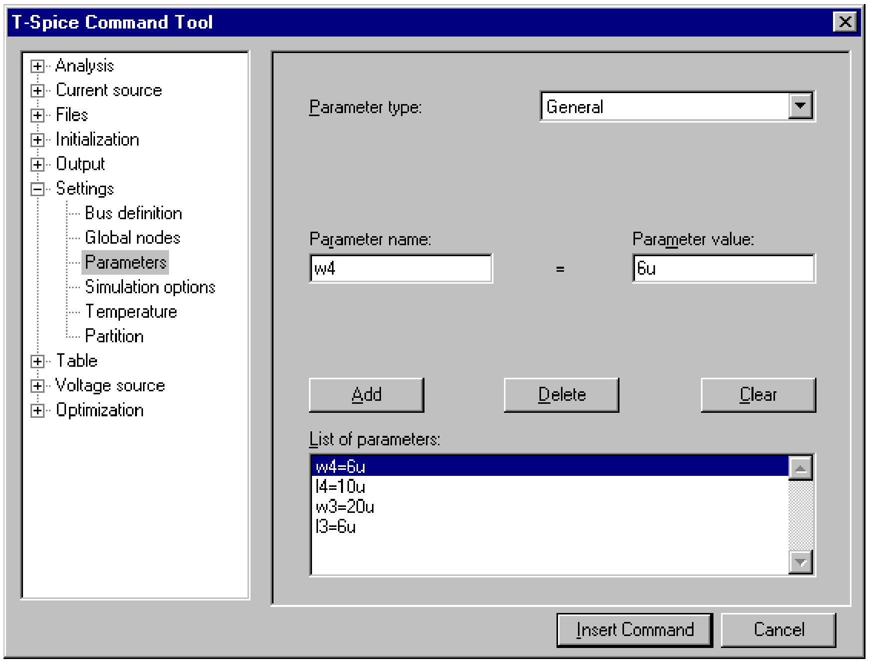
☑ Place the cursor at the beginning of any line, such as the line above the **.include** statement. Choose **Edit > Insert Command** or click the Insert Command icon in the Command toolbar:

☑ In the left-hand pane of the **Command Tool**, click **Settings**. In the right-hand pane, click the **Parameters** button.

☑ With **Parameter type** set to **General**, enter the following values. For each value pair that you add, click **Add**. T-Spice will add the parameter to the list.

Parameter name	*Parameter value*
l3	**6u**
w3	**20u**
l4	**10u**
w4	**6u**

When you have finished, the **Command Tool** will look like this:



Parameter type: General

Parameter name: w4 = Parameter value: 6u

List of parameters:

w4=6u
l4=10u
w3=20u
l3=6u

☑ Click **Insert Command**. T-Spice will insert the following line into the netlist:

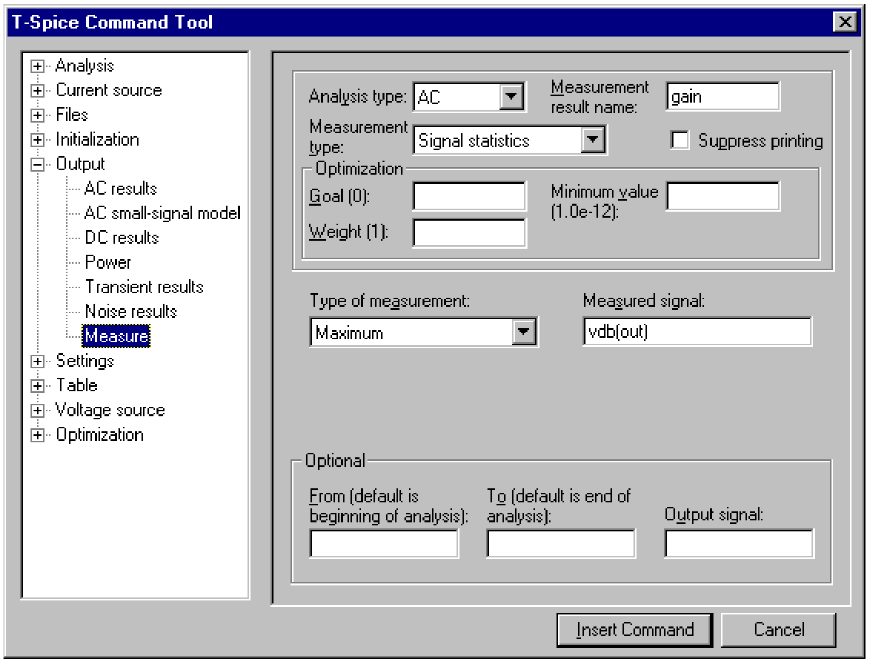
```
.param l3=6u w3=20u l4=10u w4=6u
```

☑ Next, edit the lengths and widths of output transistors **mp3** and **mn4** to optimize the results. For **mp3**, replace the existing length and width values with parameter names **l3** and **w3**, respectively. Enclose these parameter names in single quotes (‘). For **mn4**, replace the existing length and width values with parameter names **l4** and **w4**, respectively.

☑ Next, add a **.measure** command. Place the cursor at the beginning of any line and choose **Edit > Insert Command** or click the **Insert Command** icon. Double-click **Output**, then click **Measure** under **Output** (or click the **Measure** button). The **Command Tool** will appear in a form suitable for entering a **.measure** command. Type or select the following values:

Field	*Value*
Analysis type:	**AC**
Measurement result name	**gain**
Measurement type	**Signal statistics**
Type of measurement	**Maximum**
Measured signal	**vdb(out)**

☑ Leave all other fields blank. The **Command Tool** dialog will look like this:



Analysis type: AC
Measurement result name: gain
Measurement type: Signal statistics
□ Suppress printing
Optimization
Goal (0):
Weight (1):
Minimum value (1.0e-12):
Type of measurement: Maximum
Measured signal: vdb(out)
Optional
From (default is beginning of analysis):
To (default is end of analysis):
Output signal:

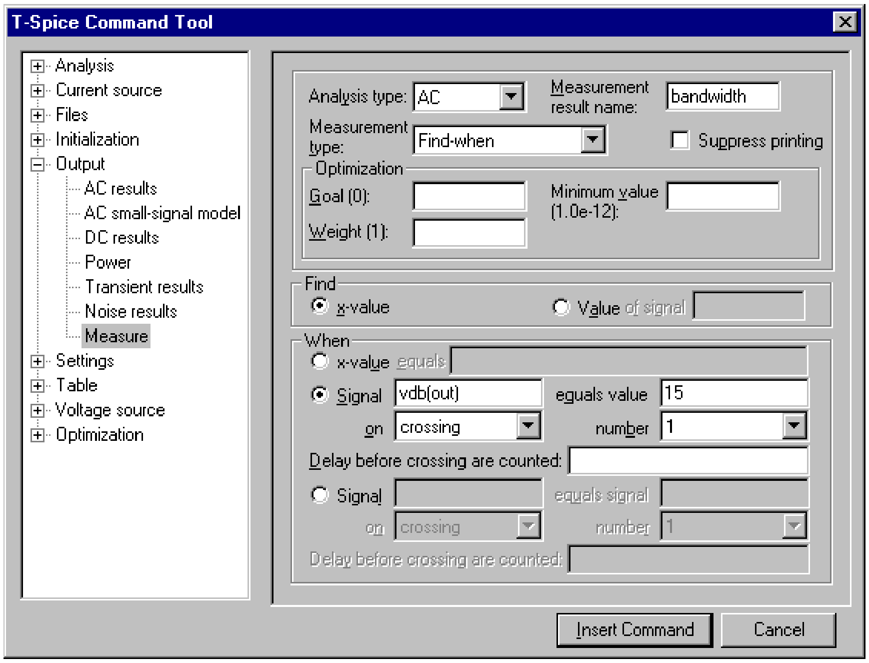
☑ Click **Insert Command** to insert the command in the netlist. T-Spice inserts the following line in the netlist:

```
.measure ac gain max vdb(out)
```

☑ For the **bandwidth** measurement, repeat the last step, using the following values:

Field	*Value*
Analysis type	**AC**
Measurement result name	**bandwidth**
Measurement type	**Find-when**
Find	**x-value**
When	**Signal**
Signal (name)	**vdb(out)**
equals value	**15**

☑ Leave all other fields blank. The **Command Tool** dialog will look like this:

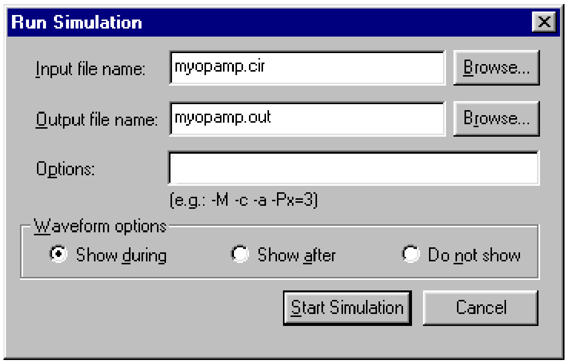


Analysis type: AC
Measurement result name: bandwidth
Measurement type: Find-when
□ Suppress printing
Optimization
Goal (0):
Minimum value (1.0e-12):
Weight (1):
Find
● x-value ○ Value of signal
When
○ x-value equals
● Signal vdb(out) equals value 15
on crossing number 1
Delay before crossing are counted:
○ Signal equals signal
on crossing number 1
Delay before crossing are counted:

☑ Click **Insert Command**. T-Spice inserts the following line in the netlist:

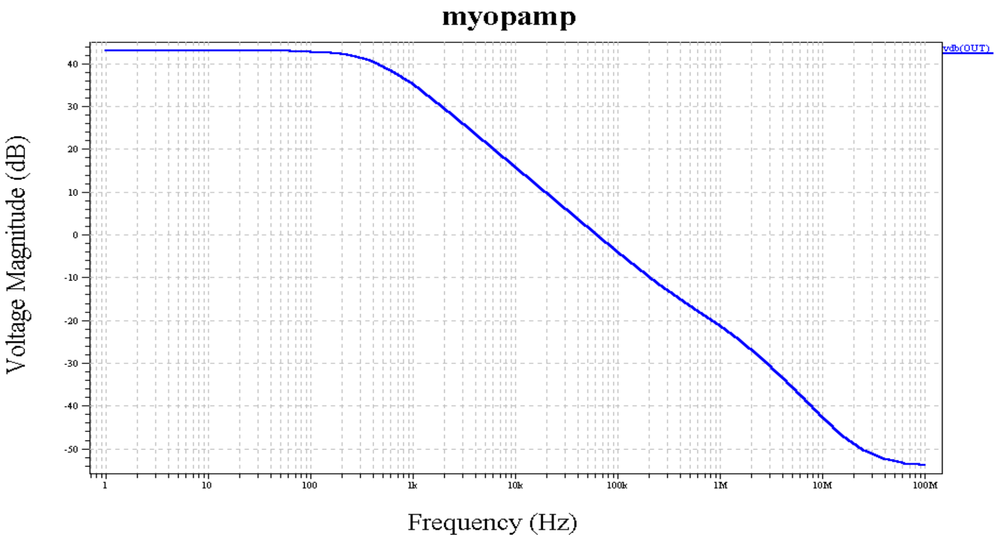
```
.measure ac bandwidth when vdb(out)=15 cross=1
```

☑ Now, run the simulation. Choose **Simulation > Run Simulation**, press **F5**, or click the Run Simulation icon in the Simulation toolbar. T-Spice displays the Run Simulation dialog.



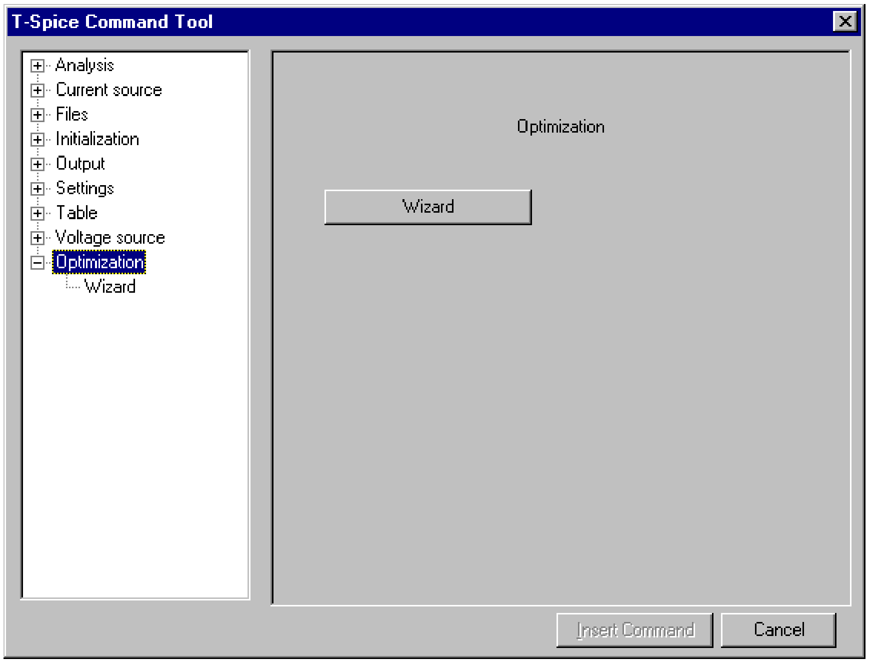
Input file name: myopamp.cir
Output file name: myopamp.out
Options:
(e.g.: -M -c -a -Px=3)
Waveform options
● Show during ○ Show after ○ Do not show

☑ Under **Waveform options**, select **Show during**, then click **Start Simulation**. T-Spice will simulate the circuit, then invoke W-Edit to display the following waveform:

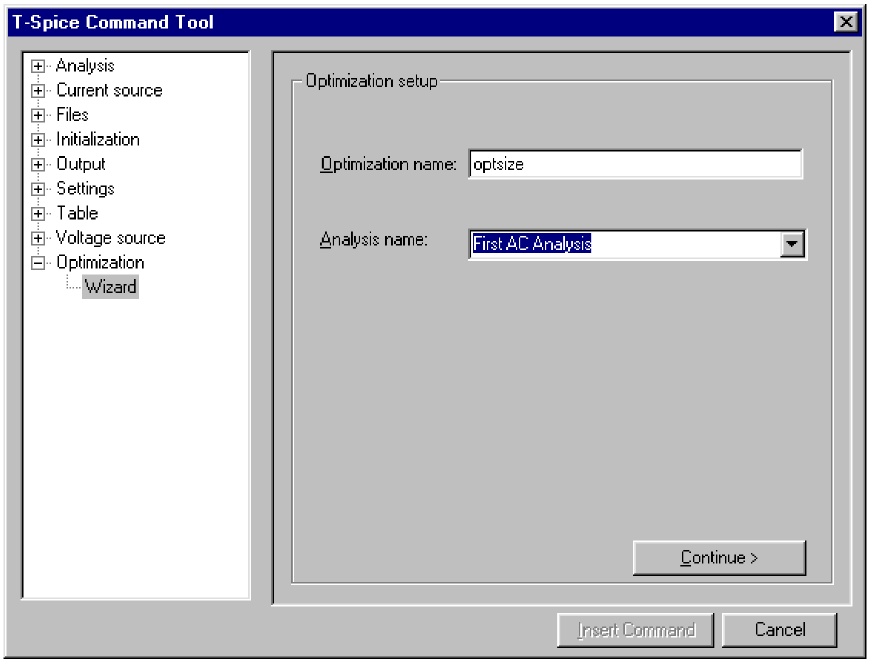


Note that the amplifier has a gain of 43.1 dB and a bandwidth of 11.4 kHz. In the following procedure, we will use the T-Spice optimization feature to modify this design to achieve a gain of 20 dB while maximizing bandwidth.

☑ Switch back to T-Spice. Choose **Edit > Insert Command** again. T-Spice displays the **T-Spice Command Tool**. In the left-hand tree, double-click **Optimization**; the **Wizard** will appear underneath.



☑ Click **Wizard** (or the **Wizard** button) to go to the first dialog, **Optimization setup**. Type **optsize** in the **Optimization name** field and type or select **First AC Analysis** as the analysis name.



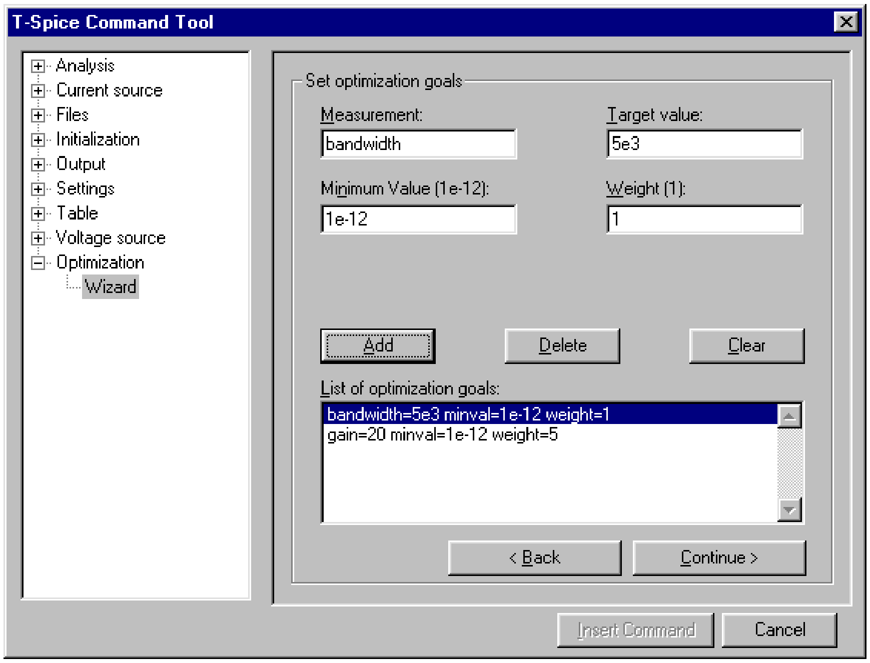
Optimization setup

Optimization name: optsize

Analysis name: First AC Analysis

☑ Click **Continue** to go to the next dialog, **Set optimization goals**. Enter **gain** for the first measurement name, with a target value of **20** (db) and a weight of **5**. Click **Add** to add this in the **List of optimization goals**.

☑ Repeat these steps to define the second measurement name **bandwidth**, target value **5e3** and weight **1**, and add them to the **List of optimization goals**. The **T-Spice Command Tool** dialog will look like this:



Set optimization goals

Measurement: bandwidth

Target value: 5e3

Minimum Value (1e-12): 1e-12

Weight (1): 1

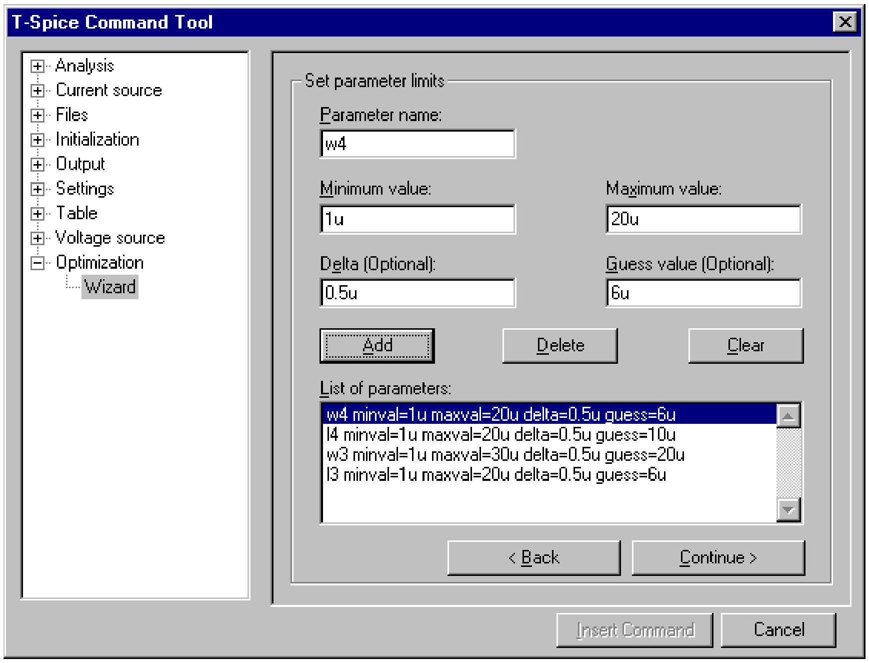
List of optimization goals:

bandwidth=5e3 minval=1e-12 weight=1
gain=20 minval=1e-12 weight=5

☑ Click **Continue** to go to the next dialog, **Set parameter limits**. First set an optimization goal for **l3**, using the first column of values in the following table:

Field	*Values*			
Parameter name	**l3**	w3	l4	w4
Minimum value	1u	1u	1u	1u
Maximum value	20u	30u	20u	20u
Delta	**0.5u**	0.5u	0.5u	0.5u
Guess value	**6u**	20u	10u	6u

☑ Click **Add** to add these values to the **List of optimization goals**. The optimization goals for **w3**, **l4**, and **w4** can be set in the same way. When you finish, the **T-Spice Command Tool** dialog will look like this:



Parameter name: w4

Minimum value: 1u

Maximum value: 20u

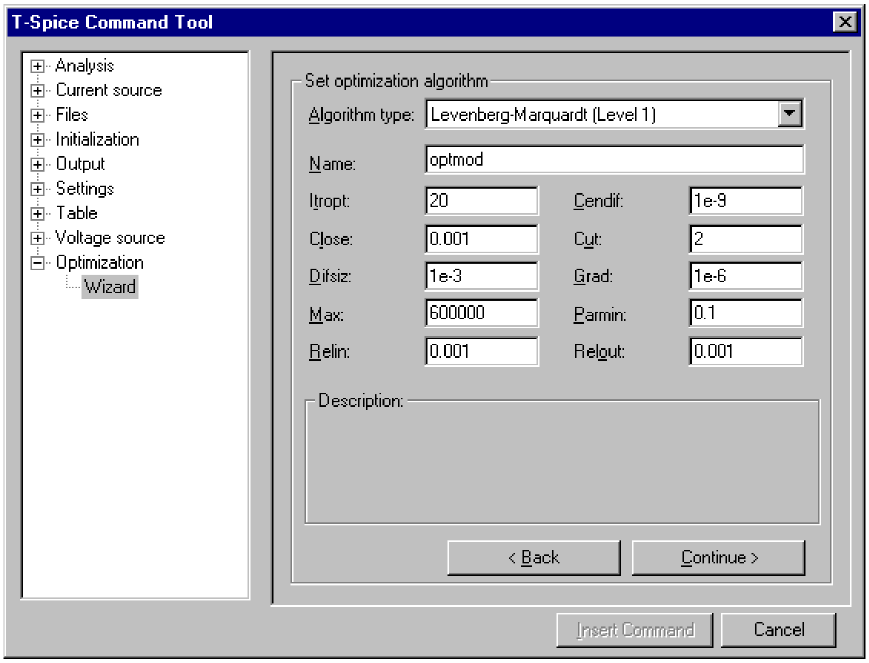
Delta (Optional): 0.5u

Guess value (Optional): 6u

List of parameters:

```
w4 minval=1u maxval=20u delta=0.5u guess=6u
l4 minval=1u maxval=20u delta=0.5u guess=10u
w3 minval=1u maxval=30u delta=0.5u guess=20u
l3 minval=1u maxval=20u delta=0.5u guess=6u
```

☑ Click **Continue** to go to the next dialog, **Set optimization algorithm**. In the **Name** field, type **optmod**. For all other values, accept the defaults. The **T-Spice Command Tool** dialog will look like this:



Set optimization algorithm

Algorithm type: Levenberg-Marquardt (Level 1)

Name: optmod

Itropt: 20 Cendif: 1e-9

Close: 0.001 Cut: 2

Difsiz: 1e-3 Grad: 1e-6

Max: 600000 Parmin: 0.1

Relin: 0.001 Relout: 0.001

Description:

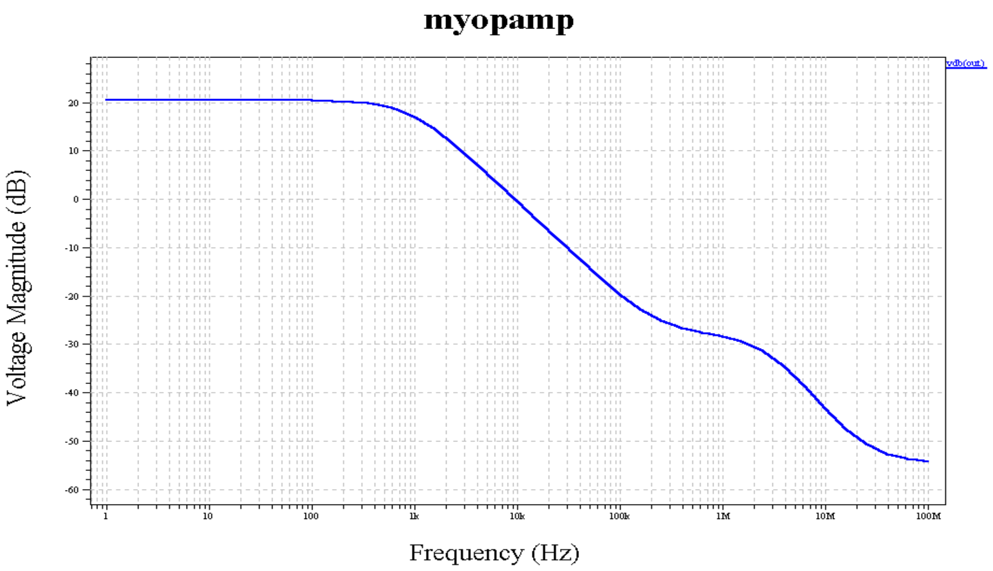
☑ Click **Continue** to go to the next dialog, **Insert command.** T-Spice displays your optimization commands in the dialog. Check them to make sure they are correct.

```
.optimize optsize model=optmod analysisname=ac
.paramlimits optsize l3 minval=1u maxval=20u delta=0.5u guess=6u
.paramlimits optsize w3 minval=1u maxval=30u delta=0.5u guess=20u
.paramlimits optsize l4 minval=1u maxval=20u delta=0.5u guess=10u
.paramlimits optsize w4 minval=1u maxval=20u delta=0.5u guess=6u
.optgoal optsize gain=20 minval=1e-12 weight=5
.optgoal optsize bandwidth=5e3 minval=1e-12 weight=1
.model optmod opt cendif=1e-9 close=0.001 cut=2 difsiz=1e-3 grad=1e-6
+      itropt=20 max=600000 parmin=0.1 relin=0.001 relout=0.001
```

☑ If you need to change a line, click **Back** to make changes.

☑ Click **Insert Command**. T-Spice inserts the optimization commands in the text editor.

☑ Rerun the simulation. This time you will see the following waveform:



Output

The list of optimization model parameters is summarized in the output file, followed by the optimization results and the optimized parameter values for **l3**, **w3**, **l4**, and **w4**.

```
* BEGIN NON-GRAPHICAL DATA

Optimization model parameters:
  Level=1
  cendif=1e-009
  close=0.001
  cut=2
  difsiz=0.001
  grad=1e-006
  itropt=20
  max=600000
  parmin=0.1
  relin=0.001
  relout=0.001

* END NON-GRAPHICAL DATA


* BEGIN NON-GRAPHICAL DATA

Optimization results:
                   Residual = 0.359805
              Gradient norm = 0.363615
        Marquardt parameter = 16.384
       Function evaluations = 45
       Number of iterations = 16

Optimized parameter values:
   l3 =  9.5000e-006
   w3 =  1.9000e-006
   l4 =  3.5000e-006
   w4 =  1.2000e-006
```

```
* END NON-GRAPHICAL DATA
```

AC analysis results are reported in different sections for different optimization runs. In this example, only one optimization run—**optsize**—was performed.

```
AC ANALYSIS - OPTIMIZE=optsize
Frequency<Hz>   vdb(out)<dB>
   1.00000e+000  2.0533e+001
   1.58489e+000  2.0533e+001
   2.51189e+000  2.0533e+001
   3.98107e+000  2.0533e+001
   6.30957e+000  2.10533e+001
...
```

The measurement results are reported at the end of each section. Because of the difference in relative weight, **gain** (weight=**5**) has higher priority in the optimization than **bandwidth** (weight=**1**), and **gain** is much closer to its optimization goal.

```
gain = 2.0533e+001
       At = 1.0000e+000

bandwidth = 1.4642e+003
```

T-Spice also supports HSPICE-compatible optimization commands, as the following example shows.

Example 4: Optimization Using HSPICE-Compatible Commands

Input	**opamp5.cir**
Output	**opamp5.out**

Input

```
* Circuit: opamp5.sp
*

* Main circuit: opamp5
.ac DEC 5 1 100MEG sweep optimize=optsize
+ results=gain bandwidth model=optmod
.model optmod opt level=1 itropt=20
.param l4=optsize(10u, 1u, 20u,0.5u) l3=optsize(6u, 1u, 20u,0.5u)
+ w4=optsize(6u, 1u, 20u,0.5u) w3=optsize(20u, 1u, 30u, 0.5u)
.measure ac gain max vdb(out) goal=20 weight=5
.measure ac bandwidth when vdb(out)=15 goal=5e3
ccomp vf1 out 2pF
cout out Gnd 2pF
mn1 vn1 vbias Gnd Gnd nmos L=10u W=6u
mn2 vm1 in1 vn1 Gnd nmos L=6u W=6u
mn3 vf1 in2 vn1 Gnd nmos L=6u W=6u
mn4 out vbias Gnd Gnd nmos l='l4' w='w4'
.include ml2_125.md
mp1 vm1 vm1 Vdd Vdd pmos L=6u W=6u
mp2 vf1 vm1 Vdd Vdd pmos L=6u W=6u
mp3 out vf1 Vdd Vdd pmos l='l3' w='w3'
.print ac vdb(out)
vbias vbias Gnd 0.8
Vdd Vdd Gnd 5.0
```

```
vdiff in2 in1 -0.0007 AC 1.0 90
vin1 in1 Gnd 2.0
* End of main circuit: opamp5
```

Output

The output of this example is the same as the output Example 3: Optimization on page 495.

13 Commands

Edit > Find

Prompts for text (the *target string*) to be searched for in the active window. The **Find Item** dialog closes when you initiate a search.

Find what

The target string to search for. Use the ▶ button to insert the special character codes used to search for a manual line break, tab break, white space, or the carat (**^**) character.

▶

Opens a submenu of special character codes that can be inserted in the target string. These codes are prefixed by the caret character (**^**) unless they are UNIX-style regular expressions.

Match whole word only

T-Spice searches only for whole words that match the specified search string.

Match case

T-Spice performs a case-sensitive search.

Regular expression

Unix-style regular expressions can be used in the target string. The **Match whole word only** option is not available in **Regular expression** mode.

Direction

- **Up**—searches backward in the active window.
- **Down**—searches forward in the active window.

Find Next

Finds the next occurrence of the target string in the active window and closes the dialog.

Replace

Opens the **Edit > Replace** (page 507) dialog.

Edit > Incremental Search

Prompts for text (the *target string*) to be searched for in the active window *as it is being typed*. In an incremental search, each character added to the target string initiates a new search.

Find what

The target string to search for. Use the ▶ button to insert the special character codes used to search for a manual line break, tab break, white space, or the carat (**^**) character.

▶

Opens a submenu of special character codes that can be inserted into the target string. These codes are prefixed by the caret character (**^**).

Search

Match case—if this is checked, only text that matches the case of the target string exactly will be found.

Direction

- **Up**—searches backward in the active window.
- **Down**—searches forward in the active window.

Find Next

Finds the next occurrence of the target string in the active window.

Replace

Opens the **Edit > Replace** (page 507) dialog.

Edit > Replace

Prompts for text to be searched for (the *target string*) and replaced (with the *replace string*) in the active window.

Find what

The target string to search for. Use the ▶ button to insert the special character codes used to search for a manual line break, tab break, white space, or the carat (**^**) character.

▶

Opens a submenu of special character codes that can be inserted into the target string. These codes are prefixed by the caret character (**^**).

Replace with

The replace string, which can include character codes from the ▶ submenu.

Search

Options for limiting the search. Check an option to make it active.

- **Match whole word only**—only an exact match to the target string will be found.
- **Match case**—only text that matches the case of the target string exactly will be found.
- **Regular expression**—Unix-style regular expressions can be used in the target string. The **Match whole word only** option is not available in **Regular expression** mode.

Direction

- **Up**—searches backward in the active window.
- **Down**—searches forward in the active window.

Find Next

Finds the next occurrence of the target string in the active window.

Replace

If a target string is selected, replaces it with the replace string.

Replace All

Replaces *all* occurrences of the target string in the active window with the replace string.

Edit > Goto Line

Prompts for a line number, then places the cursor at the beginning of the corresponding line in the active window.

Edit > Insert Command

Opens the **T-Spice Command Tool**, which automates the insertion of T-Spice commands and device statements, in correct SPICE format, into the active window.

Commands are inserted into the active window to the right of the cursor position or replacing highlighted text.

Use the left side of the dialog to choose a command, and the right side to enter parameters.

Left side

Displays a hierarchical list of command categories. Double-click a category (or click the plus or minus sign next to the category name) to expand or collapse the list of specific commands.

Right side

When a specific command is highlighted, this side of the dialog contains a field for each of the variables required by the command.

Insert Command

Inserts the command, with the specified options and arguments, into the active window. Commands are inserted above the current line if nothing is selected and below the current line if something is selected.

View > Simulation Status

Toggles the visibility of the **Simulation Status** window.

The **Simulation Status** window shows simulation statistics and progress information, as well as any warnings or error messages, for the currently running or most recently completed simulation

Input file

Name of the input file.

Output file

Name of the output file.

Progress

Displays the type of operation currently in progress, its duration in nanoseconds, and the percentage of the simulation completed.

Total nodes

Total number of nodes simulated.

Total devices

Total number of devices simulated.

Active devices

Total number of active devices simulated.

Passive devices

Total number of passive devices simulated.

Independent sources

Total number of independent sources simulated.

Controlled sources

Total number of controlled sources simulated.

View > Simulation Manager

Toggles the visibility of the **Simulation Manager**. This window may be displayed using a docked or undocked view. (For details about docking and undocking, see Simulation Manager on page 10.)

The **Simulation Manager** controls all T-Spice simulations. It queues files for simulation, displays their processing status, and allows you to stop or pause a simulation. You can also highlight a file in this window to view simulation results in the **Simulation Status** window.

Each simulation occupies one row, and each row has six attributes:

Status — Simulation status. Possible states include:

- **Queued**—the simulation is in the queue and will run when the simulation engine is available.
- **Running**—the simulation is underway.
- **Paused**—simulation has been suspended by the user.
- **Finished**—the simulation has run and output is available.
- **Stopped**—the simulation was stopped by the user.
- **Failed**—the simulation has failed to run.

Input file — Full pathname of the input file

Output file — Full pathname of the output file

Options — Lists all command-line options entered in the **Command Options** field of the **Run Simulation** dialog. For further information, see **Simulation > Run Simulation** (page 511) and Command-Line Options on page 28.

Start Date/Time — The date and time at which the simulation began execution.

Elapsed Time — The total simulation run time in **hh:mm:ss** format. Pausing a simulation will not interrupt the measurement of elapsed time.

When the Simulation Manager is *undocked*, the following buttons are available to control queued simulations:

Run Simulation — Invokes **Simulation > Run Simulation** (page 511). If a file is highlighted, the appropriate file and path names will automatically populate the **Input file** and **Output file** fields.

Stop — Stops simulation processing for the highlighted file. Once a simulation is stopped, it cannot be resumed.

Pause/Resume — Pauses simulation when the status of the highlighted file is **Running**, and resumes simulation when the status of the highlighted file is **Paused**.

Delete — Removes the highlighted file from the simulation queue. If the simulation is running or paused, you will be prompted to stop processing on the file before it is deleted.

Empty List — Removes all files from the simulation queue. If any simulations are running or paused, you will be prompted to stop processing before they are deleted.

Show Waveform — Invokes the W-Edit waveform viewer for the selected simulation.

Show Netlist — Opens a window with the selected input file. If the selected file is already open, makes that window active.

Show Output — Opens a window with the output file (**.out**) corresponding to the selected simulation. If already open, makes that window active.

Right-clicking in any field of the Simulation Manager (either docked or undocked) opens a pop-up menu with the options **Run Simulation**, **Delete**, **Empty List**, **Show Waveform**, **Show Netlist**, and **Show Output**, as described above. Two additional menu items specify the window's visibility (**Hide**) and docking (**Docking view**).

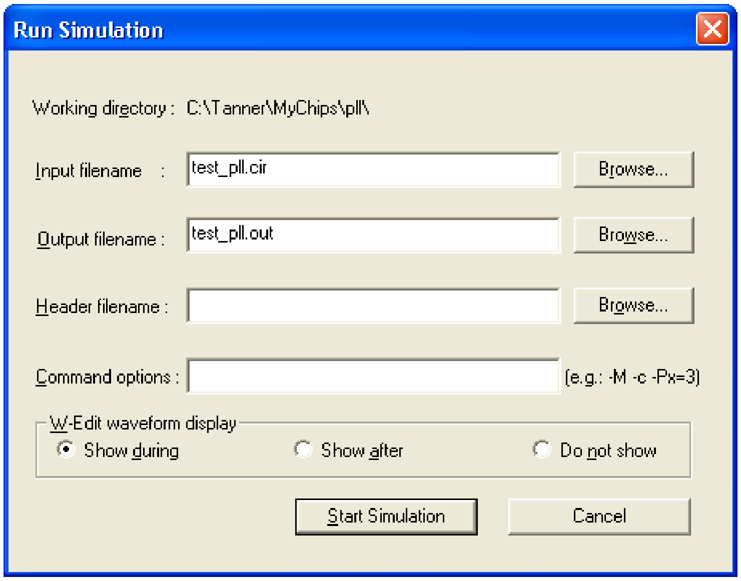
Note: The simulation-specific options **Delete**, **Show Waveform**, **Show Netlist**, and **Show Output** are only enabled when you right-click on a simulation entry.

View > Show Waveform Viewer

Opens W-Edit. If the active window is a file that has been previously simulated and the file is still in the **Simulation Manager** queue, the corresponding **.out** file will be displayed when W-Edit opens.

Simulation > Run Simulation

Opens the **Run Simulation** dialog used to specify filenames and command-line options before launching a T-Spice simulation.



Run Simulation

Working directory : C:\Tanner\MyChips\pll\

Input filename : test_pll.cir

Output filename : test_pll.out

Header filename :

Command options :

W-Edit waveform display: ● Show during ○ Show after ○ Do not show

Working Directory

Displays the directory path to the file currently selected as the **Input Filename**, below.

Input Filename

The name of the input netlist to be simulated. You can select **Browse** to navigate to the simulation file, or type the filename directly in the editing field.

To specify a file contained in the current **Working Directory** (above), simply type the filename in the editing field. To change the Working Directory, enter the filename with its full directory path; T-Spice will update the **Working Directory** automatically.

Output file name

The name of the simulation output file. You can type a name directly in the editing field or select **Browse** to navigate to another file. To specify an output file outside the **Working Directory**, include the full directory path.

The default value is equal to the input file name with a **.out** extension (***input_name*.out**).

Note: You can instruct T-Spice to automatically accept the output filename, or to first check if the file exists and then ask permission to overwrite it. This setting is controlled in .

Header Filename	Name of a header file to be included in the T-Spice simulation. You can type a name directly in the editing field or select **Browse** to navigate to another file. To specify a header file outside the **Working Directory**, include the full directory path. This option is equivalent to copying the contents of the header file to the beginning of your input netlist.
Command Options	Use this field to enter command-line options, which modify a simulation without altering the input file. You can enter as many options as desired, separating each with a space. Refer to Command-Line Options on page 28 for a description of available options and their proper syntax.
W-Edit waveform display	Options for displaying simulation progress in W-Edit. Check an option to make it active. ▪ **Show during**—displays traces in W-Edit while the simulation is running. ▪ **Show after**—displays traces in W-Edit once the simulation is complete. ▪ **Do not show**—does not open W-Edit.
Start simulation	Adds the file to the end of the **Simulation Manager** queue.

Simulation > Batch Simulations

Opens the **Create Batch** dialog used to list input files that will be added simultaneously to the **Simulation Manager** queue.

Add Simulation	Opens the **Add Simulation** dialog to add a new file to the list of files to be submitted to the Simulation Manager queue.
Remove Simulation	Removes the selected file from the list of files to be submitted to the Simulation Manager queue.
Remove All Simulations	Removes all files from the list of files to be submitted to the Simulation Manager queue.
Submit	Sends the files listed in the **Create Batch** dialog to the **Simulation Manager** queue.

Add Simulation

Fields in the **Add Simulation** dialog fields are the same as fields in the **Simulation > Run Simulation** (page 511) dialog.

Table > Generate Table

Prompts for the basename of a set of external table files to be generated from the (active) input file.

The input file must contain at least one model and one device (MESFET, MOSFET, JFET, or diode) definition. It may also contain table control commands such as **.vrange** (page 140) and **.gridsize** (page 69), for customizing generated tables. The generated tables can be used with the **.table** (page 133) command as device models.

Table file basename A table file basename related to the input file name is automatically suggested.

ASCII Text Generate tables in ASCII (human-readable) format (**.ftx**, **.qtx** format).

Binary Generate tables in binary (machine-readable) format (**.f**, **.q** format).

Table > Evaluate Table

Prompts for terminal voltages to use when evaluating an external table generated from the (active) input file.

The input file must contain at least one model and one device (MESFET, MOSFET, JFET, or diode) definition. It may also contain table control commands such as **.vrange** (page 140) and **.gridsize** (page 69), for customizing the generated table. The generated table is evaluated at the specified terminal voltages. Current and charge values corresponding to the specified voltages are reported in the **Simulation Status** window.

Terminal … **voltage** Input voltages for each terminal.

Table > Convert Table

Prompts for the name and path of an external table file to be converted from ASCII (**.ftx**, **.qtx**) to binary (**.f**, **.q**) format or vice versa. Files that have been converted are added to the **Simulation Manager** queue and can be viewed in the **Simulation Status** window.

File name The name of the table file to be converted. The wildcard character ***** can be used. However, only one file can be converted at a time. The scrolling window lists all files in the source directory.

Files of type Select from the menu to limit the types of files displayed.

Open The name of the table file to be converted.

Table > Monotonicity Check

Prompts for the name and path of an external table file to be checked for monotonicity. (Lack of monotonicity can cause severe convergence problems during simulation.)

File name

The name of the table file to be checked. The wildcard character ***** can be used. However, only one file can be checked at a time. The scrolling window lists all files in the source directory.

Files of type

Select from the menu to limit the types of files displayed.

Open

The name of the table file to be checked.

When you select a file for checking, T-Spice opens the **Save Output to File** dialog, prompting you for the name and path of an output file.

Output file name

An output file name (with a **.prt** extension) based on the input file name is automatically suggested.

Table > Print Table

Prompts for the name and path of an external table file to be printed to a **.prt** format file in user-readable form.

File name

The name of the table file to be printed. The wildcard character ***** can be used. However, only one file can be printed at a time. The scrolling window lists all files in the source directory.

Files of type

Select from the menu to limit the types of files displayed.

Open

The name of the table file to be printed.

When you select a file for printing, T-Spice opens the **Save Output to File** dialog, prompting you for the name and path of an output file.

Output file name

An output file name (with a **.prt** extension) based on the input file name is automatically suggested.

14 References

The following references may be consulted for further information on various aspects of circuit design, modeling, and simulation.

[1] P. Antognetti and G. Massobrio (eds.), *Semiconductor Device Modeling with SPICE*, 2nd ed. New York: McGraw-Hill, 1993.

[2] M. A. Belkerdid and P. F. Wahid, "Rise time and frequency correlation for crosstalk in high-speed packaging," *IEPS Proc. Tech. Conf.* (Dallas), pp. 617–636, November 1988.

[3] F. H. Branin, Jr., "Transient analysis of lossless transmission line," *Proc. IEEE (Letters)*, vol. 55, pp. 2012–2013, 1967.

[4] T. Dhaene and D. De Zutter, "Selection of lumped element models for coupled lossy transmission lines," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 7, pp. 805–815, July 1992.

[5] D. Divekar, "Comments on GaAsFET Device and Circuit Simulation in SPICE," *IEEE Trans. Electron Devices*, vol. ED-34, no. 12, pp. 2564–2565, December 1987.

[6] A. R. Djordjevic, T. K. Sakar, and R. G. Harrington, "Time-domain response of multiconductor transmission lines," *Proc. IEEE*, vol. 75, pp. 743–764, June 1987.

[7] J. H. Huang, Z. H. Liu, M. C. Jeng, K. Hui, M. Chan, P. K. Ko, and C. Hu, *BSIM3 Manual (Version 2.0).* Berkeley, CA: University of California, 1994.

[8] P. K. Ko, C. Hu, et al., *BSIM3v3 Manual (Final Version).* Berkeley, CA: University of California, 1995.

[9] M. A. Maher, *A Charge-Controlled Model for MOS Transistors.* Department of Computer Science Document 5223:TR:86. Pasadena, CA: California Institute of Technology, 1989.

[10] M. A. Maher and C. Mead, "A Physical Charge-Controlled Model for MOS Transistors," in *Advanced Research in VLSI*, P. Losleben (ed.). Cambridge, MA: MIT Press, 1987, p. 211.

[11] C. Mead, *Analog VLSI and Neural Systems.* Reading, MA: Addison-Wesley, 1989.

[12] L. W. Nagel, *SPICE2: A Computer Program to Simulate Semiconductor Circuits.* Electronics Research Laboratory Memorandum ERL-M520. Berkeley, CA: University of California, 1975.

[13] I. W. Smith, H. Statz, A. Hans, and R. A. Pucel, "On Charge Nonconservation in FETs," *IEEE Trans. Electron Devices*, vol. ED-34, no. 12, pp. 2565–2568, December 1987.

[14] H. Statz et al., "GaAsFET Device and Circuit Simulation in SPICE," *IEEE Trans. Electron Devices*, vol. ED-34, no. 3, pp. 160–169, February 1987.

[15] Y. P. Tsivids, *Operation and Modeling of the MOS Transistor.* New York: McGraw-Hill, 1987.

[16] A. Vladimirescu, *The SPICE Book.* New York: Wiley & Sons, 1994.

[17] D. E. Ward, *Charge-Based Modeling of Capacitance in MOS Transistors*. Integrated Circuit Laboratory Report G201-11. Stanford, CA: Stanford University, 1981.

[18] D. E. Ward and R. W. Dutton, “A Charge-Oriented Model for MOS Transistor Capacitances,” *IEEE J. Solid State Circuits*, SC-13, 1978.

[19] W. Liu et al., *BSIM3v3.2.2 MOSFET Model Users Manual*, Berkeley, CA: University of California, 1999.

[20] W. Liu, X. Jin, et. al., *BSIM3v3.2.2 MOSFET Model User’s Manual*. Berkeley, CA: University of California, 1999.

[21] Matthias Bucher, Christophe Lallement, et. al., *The EPFL-EKV MOSFET Model Equations for Simulation Technical Report*. Lausanne, Switzerland: Electronics Laboratories, Swiss Federal Institute of Technology (EPFL), 1998.

[22] P. Su, H. Wan, Z. H. Liu, S. Fung, et. al., *BSIMSOI3.1 MOSFET Model Users’ Manua*l. Berkeley, CA: University of California, 2003.

[23] X. Xi, M. Dunga, et. al., *BSIM4.4.0 MOSFET Model User’s Manual*. Berkeley, CA: University of California, 2004.

Index

Credits

Software Development

Ken Van de Houten	Dan'l Leviton
Alex Ivanov	

Quality Assurance

Luba Gromova	Lena Neo
Ken Van de Houten	

Documentation

Judy Bergstresser	Ken Van de Houten

Additional Credits

T-Spice is based on version 5.0 of CAzM (Circuit AnalyZer with Macromodeling) written by Donald J. Erdman and Donald J. Rose Gary B. Nifong, Bill Richards, Stephen Kenkel and Ravi Subrahmanyan MCNC Center for Microelectronics and Duke University.

T-Spice contains object modules from the Intel Math Kernel Library (Intel MKL) package, Copyright © 2002-2005, Intel Corporation.

T-Spice uses the Sparse1.3 matrix library, copyright © 1985-1988 by Kenneth S. Kundert, University of California, Berkeley.

T-Spice uses the SuperLU sparse linear solver library, copyright © 2003, The Regents of the University of California.

BSIM models BSIM4, BSIM3v322 and BSIMSOI are based upon software developed by the BSIM Research Group, University of California, Berkeley, ©1990-2004, Regents of the University of California.

EKV models are based on the EPFL-EKV MOSFET Report by the Electronics Laboratories, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, ©1998.

Philips models are based upon software developed by Philips Electronics N.V. ©2005

SiMKit is a trademark of Philips Electronics N.V.

The following copyright notice is applicable to Sparse, SuperLU, and BSIM: