MODELngspicer User Guide

ペ E

September 28, 2025

Contents

1 Introduction

MODELngspicer is a GUI application built on top of ngspice, designed to help circuit design optimization and device modeling. The graphical interface is developed using Qt library (PySide6), allowing users to adjust circuit parameters intuitively and visualize simulation results in real time. This seamless integration between control and feedback enables rapid iteration and deeper insight into circuit behavior.

Ngspice (https://ngspice.sourceforge.io) is an open-source SPICE circuit simulator that supports a wide range of analyses, including DC, AC, transient, noise, distortion, and S-parameter analysis. It also accommodates industry-standard models such as Verilog-A and BSIM, enabling high-precision device modeling. Since it operates via command line, ngspice is well suited for complex scripted computations and automated analysis workflows.

2 Installation

2.1 System Requirements

MODELngspicer has been tested under the following environment:

- Operating system: Windows 10 or later (64-bit)
- Python: Version 3.13 or later
- Required packages: PySide6, PyQtGraph, NumPy
- Ngspice: Version 41 or later

2.2 Installation on Windows

This section outlines the installation procedure using Chocolatey, a package manager for Windows.

1. Install Chocolatey
   Open PowerShell as Administrator and run the following command. For more details, visit Chocolatey's official installation page (https://chocolatey.org/install).

   ```
   Set-ExecutionPolicy Bypass -Scope Process -Force; `
   [System.Net.ServicePointManager]::SecurityProtocol = `
   [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `
   iex ((New-Object System.Net.WebClient).DownloadString(`
   'https://community.chocolatey.org/install.ps1'))
   ```

2. Install Python and Ngspice
   Run the following commands in an administrator PowerShell to install Python and Ngspice. After installation, verify that `python`, `pip`, and `ngspice_con` commands are available.

```
choco install python
choco install ngspice
```

3. Install required packages
   Run the following command to install required Python libraries.

   ```
   pip install pyside6 pyqtgraph numpy
   ```

4. Launch the application
   Once the setup is complete, you can launch MODELngspicer by double-clicking `run.vbs` or `run.bat` located in the project folder.

3 GUI and Basic Operations

3.1 Interface Overview

Figure 1 shows the graphical interface layout.

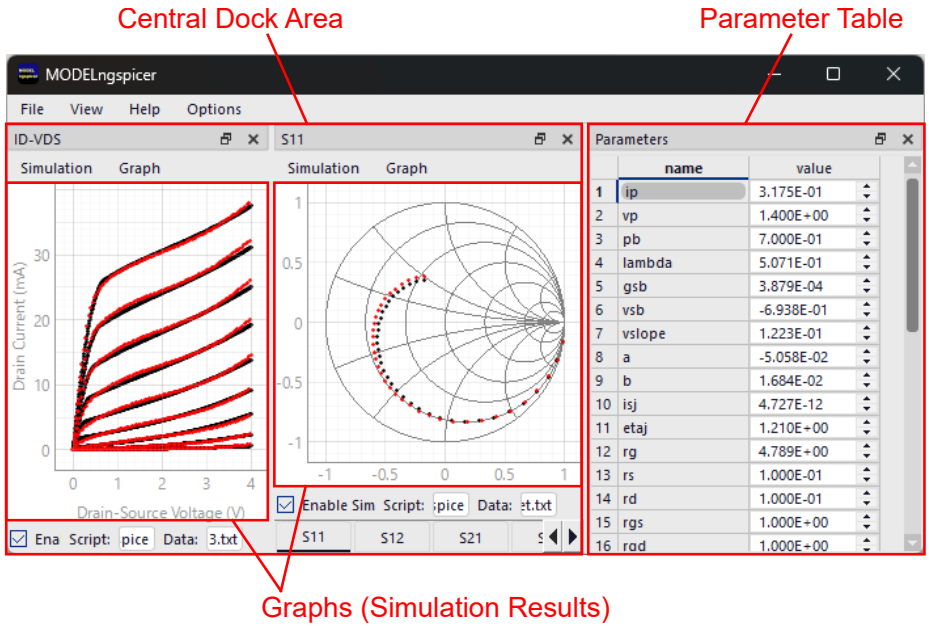

Figure 1: Graphical interface

- **Central Dock Area**
  The area where the simulation widgets are stored. Up to 10 pages can be added.
- **Parameter Table**
  Displays a list of parameters loaded from "File > Import Params...". Each parameter value is shown in a spin box and can be adjusted using a mouse wheel.
- **Graph**
  Simulation results and measurement data are plotted. Simulation results are shown as black dots, while measurement data as red dots.

3.2 Operation Flow

The user needs to prepare at least the following two files. The formats of these files are explained in the next section.

- Parameter definition file
- Ngspice script file

The basic operation workflow is outlined below.

1. **Import parameters**
   Select a parameter definition file via "File > Import Params...". The loaded parameters will be displayed in the parameter table.
2. **Select ngspice script**
   Select an ngspice script (.spice, .sp, or .cir) via "Simulation > Select Ngspice Script...".
3. **Select external data (optional)**
   Load external data such as measurement results or target values via "Simulation > Select Data...".
4. **Enable/Disable simulation (optional)**
   Simulation can be toggled via "Simulation > Enable/Disable Simulation" or the check box at the bottom left of the simulation widget. Disabling unused simulations may help reduce execution time.
5. **Adjust parameters**
   Adjust parameter values using the spin boxes in the parameter table.
   - Mouse wheel → fine adjustment
   - Ctrl + mouse wheel → coarse adjustment
6. **Review results**
   Simulation results and external data are plotted in the graph.
7. **Save parameters**
   Save the adjusted parameters via "File > Export Params..."

4 File Formats

4.1 Parameter Definition File

Parameters are defined in a plain text (.txt). Each line must begin with a "+" symbol as shown in Listing 1. The "+" symbol is part of SPICE syntax and indicates a continuation from the previous line. By using this format, the parameter definitions can be seamlessly expanded into `.param` or `.model` statements within SPICE scripts.

Listing 1: Example of parameter definition

```
+ is = 1e-14
+ n = 1.5
+ rs = 0.5
+ ...
```

Listing 2 shows an example of an SPICE script expanding the parameter definition into a `.param` statement. When an `.include model.txt` is placed directly below a `.param`, the contents of `model.txt` are interpreted as `.param` arguments.

Listing 2: SPICE script expanding into a `.param` statement

```
.param
.include model.txt
```

Listing 3 shows an example of an SPICE script expanding the parameter definition into a `.model` statement. In this example, the contents of `model.txt` are applied as parameters for the diode model `DIODE1`. With this approach, you can define custom device models.

Listing 3: SPICE script expanding into a `.model` statement

```
.model DIODE1 D (
.include model.txt
+ )
```

4.2 Simulation Result and Data Files

Both simulation results and external data use a common text format (.txt). The file structure is as follows and Listing 4 shows an example.

- No header
- First column: X-axis data
- Second and subsequent columns: Y-axis data
- Delimiters: Space or tab

Listing 4: Example of a data file

```
1e6 -3.2 -1.1
2e6 -3.5 -1.3
3e6 -3.8 -1.6
...
```

To export simulation results within SPICE scripts, use the `wrdata` command. The output file must have the same basename as the script file, with a .txt extension. This naming rule enables to automatically identify the output file and plot it on the graph.

Listing 5 shows an example of a script exporting simulation results. In this example, running `iv.spice` will produce `iv.txt` containing the current values of `i(v1)` and `i(v2)`.

Listing 5: SPICE script exporting simulation results

```
set wr_singlescale
wrdata iv.txt i(v1) i(v2)
```

5 Use Cases

5.1 LC Bandpass Filter Design

This section introduces an example of designing an LC bandpass filter using MODELngspicer. Figure 2 shows the schematic of the filter, with nodes labeled from `n01` to `n05`. The parameters `C1`, `L2`, `C3` and `L4` are used.

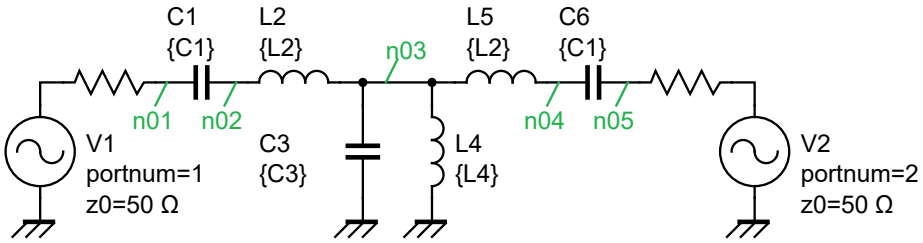

Figure 2: Schematic of the LC bandpass filter

We create a parameter file (Listing 6) and ngspice scripts (Listing 7 and 8) that calculate S parameters of the filter.

Listing 6: model_initial.txt

```
+ c1=6.234E-12
+ l2=4.104E-07
+ c3=1.826E-10
+ l4=1.401E-08
```

Listing 7: bpf1_S11.spice

```
bpf1_S11.spice
* This script performs an S-parameter analysis of a bandpass filter.

.param
.include model.txt
+ L5=L2 C6=C1

* Port definitions for S-parameter analysis
V1 n01 0 dc 0 portnum 1 z0 50
V2 n05 0 dc 0 portnum 2 z0 50

* Bandpass filter topology
C1 n01 n02 {C1}
```

```
L2 n02 n03 {L2}
C3 n03 0 {C3}
L4 n03 0 {L4}
L5 n03 n04 {L5}
C6 n04 n05 {C6}

.control
* S-parameter analysis from 50 MHz to 150 MHz with 200 points
sp lin 200 50Meg 150Meg

* Write S11 magnitude in dB to file
set wr_singlescale
wrdata bpf1_S11.txt db(s_1_1)

.endc
.end
```

Listing 8: bpf1_S21.spice

```
bpf1_S21.spice
* This script performs an S-parameter analysis of a bandpass filter.

.param
.include model.txt
+ L5=L2 C6=C1

* Port definitions for S-parameter analysis
V1 n01 0 dc 0 portnum 1 z0 50
V2 n05 0 dc 0 portnum 2 z0 50

* Bandpass filter topology
C1 n01 n02 {C1}
L2 n02 n03 {L2}
C3 n03 0 {C3}
L4 n03 0 {L4}
L5 n03 n04 {L5}
C6 n04 n05 {C6}

.control
* S-parameter analysis from 50 MHz to 150 MHz with 200 points
sp lin 200 50Meg 150Meg

* Write S21 magnitude in dB to file
set wr_singlescale
wrdata bpf1_S21.txt db(s_2_1)

.endc
.end
```

Launch the application (run.vbs or run.bat) and load `model_initial.txt` from "File > Import Params...". Next, go to "Simulation > Select Ngspice Script..." and select `bpf1_S11.spice` and `bpf1_S21.spice`.

You can modify the layout from "View > Tiling > 1 x 2" and rename the simulations from "Page 1" and "Page 2" to "S11" and "S21" respectively, via "Simulation > Rename Title". Also, you can set the X and Y axis labels via "Graph > Axis Titles".

Figure 3 shows the screenshot. Try to adjust parameters using spin boxes and check that the filter responses change in real time.

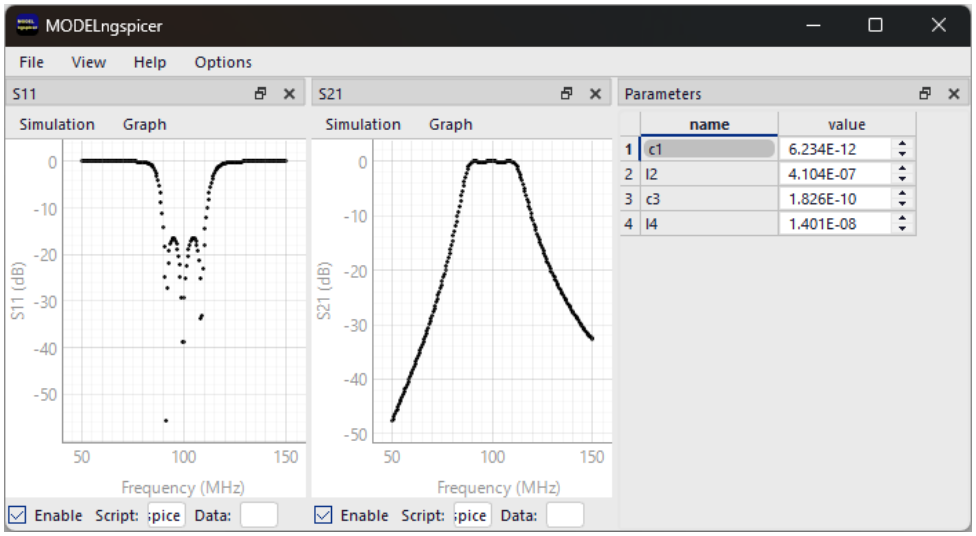

Figure 3: LC bandpass filter design

5.2 Diode Modeling

This section presents an example of diode modeling using MODELngspicer. We will adjust the model parameters based on the datasheet of Onsemi 1N4148. In this example, we focus on the following characteristics.

- Forward and reverse I-V characteristics (I_F-V_F, I_R-V_R)
- Junction capacitance vs. reverse voltage (C_T-V_R)
- Reverse recovery time (t_{rr})

5.2.1 Model Parameters

Table 1 shows a list of diode model parameters targeted for adjustment, and Listing 9 shows the parameter definition file.

Table 1: Diode Parameters

Name	Value	Unit	Description
is	4.772E-09	A	Saturation current
n	1.931E+00	-	Emission coefficient
rs	7.055E-01	Ω	Ohmic resistance
bv	1.423E+02	V	Reverse breakdown voltage
ibv	1.000E-03	A	Current at breakdown voltage
nbv	4.784E+01	-	Breakdown emission coefficient
ikf	0.000E+00	A	Forward knee current
ikr	5.000E-07	A	Reverse knee current
jtun	3.987E-11	A	Tunneling saturation current
ntun	4.914E+02	-	Tunneling emission coefficient
xtitun	3.000E+00	-	Tunneling saturation current exponential
cjo	8.695E-13	F	Zero-bias junction capacitance
fc	5.000E-01	-	Coefficient for forward-bias depletion capacitance formula
m	2.266E-02	-	Area junction grading coefficient
vj	7.000E-01	V	Junction potential
tt	1.660E-08	sec	Transit time
eg	1.110E+00	eV	Activation energy
xti	3.000E+00	-	Saturation current temperature exponent
tnom	2.700E+01	°C	Parameter measurement temperature

Listing 9: model_final.txt

```
+ is=4.772E-09
+ jsw=0.000E+00
+ n=1.931E+00
+ rs=7.055E-01
+ bv=1.423E+02
+ ibv=1.000E-03
+ nbv=4.784E+01
+ ikf=0.000E+00
+ ikr=5.000E-07
+ jtun=3.987E-11
+ ntun=4.914E+02
+ xtitun=3.000E+00
+ cjo=8.695E-13
+ fc=5.000E-01
+ m=2.266E-02
+ vj=7.000E-01
+ tt=1.660E-08
+ eg=1.110E+00
+ xti=3.000E+00
+ tnom=2.700E+01
```

5.2.2 Forward I-V

We begin by adjusting the model based on the forward I-V characteristics (I_F-V_F). Listing 10 shows the ngspice script for I_F-V_F.

Listing 10: D_FORWARD_IV.spice

```
D_FORWARD_IV.spice
* This script performs a DC sweep of forward voltage across a diode
* to extract its forward current characteristics.

* Diode model
.model DIODE1 D (
.include model.txt
+ )

* Test circuit
VF n01 0 dc 0 $ Forward voltage source
VIF n01 n02 dc 0 $ Forward current probe
D1 n02 0 DIODE1 $ Test diode

.control
option TEMP=25 $ Set simulation temperature to 25°C

* Sweep VF from 0.3 V to 1.5 V in 0.01 V steps
dc VF 0.3 1.5 0.01

* Write forward current data to file
set wr_singlescale
wrdata D_FORWARD_IV.txt i(VIF)

.endc
.end
```

Figure 4 shows a screenshot after tuning the forward I-V characteristics. Here, we adjust parameters: `is` (saturation current), `n` (emission coefficient), and `rs` (ohmic resistance). In the low-current region, `is` and `n` affect the intercept and slope of the curve, respectively. In the high-current region, `rs` becomes a dominant factor.

Red dots represent the datasheet readings, while black dots indicate the simulation results. The two sets of data are in good agreement.

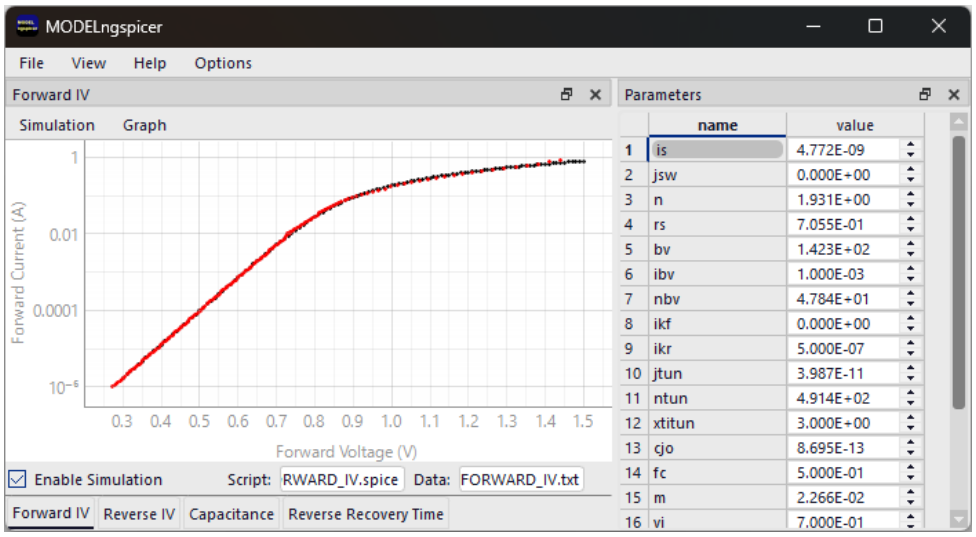

Figure 4: Forward I-V characteristics

5.2.3 Reverse I-V

Next, we adjust the model based on the reverse I-V characteristics (I_R-V_R). Listing 11 shows the ngspice script for I_R-V_R.

Listing 11: D_REVERSE_IV.spice

```
D_REVERSE_IV.spice
* This script performs a DC sweep of reverse voltage across a diode
* to extract its reverse current characteristics, including breakdown behavior
    .

* Diode model
.model DIODE1 D (
.include model.txt
+ )

* Test circuit
VR n01 0 dc 0 $ Reverse voltage source
VIR n01 n02 dc 0 $ Reverse current probe
D1 0 n02 DIODE1 $ Test diode

.control
option TEMP=25 $ Set simulation temperature to 25°C
option GMIN=4e-10 $ Set minimum conductance

* Sweep VR from 10 V to 146 V in 1 V steps
dc VR 10 146 1

* Write reverse current data to file
set wr_singlescale
wrdata D_REVERSE_IV.txt i(VIR)
```

```
25
26 .endc
27 .end
```

Figure 5 shows a screenshot after tuning the reverse I-V characteristics. Here, we adjust parameters: `jtun` (tunneling saturation current), `ntun` (tunneling emission coefficient), `bv` (reverse breakdown voltage), `nbv` (breakdown emission coefficient), `ikr` (reverse knee current), and the environment variable `GMIN` (minimum conductance).

In the low-voltage region (up to 70 V), leakage current is primarily determined by `is` and `GMIN`. Since `is` has already been tuned based on the forward I-V characteristics, only `GMIN` is adjusted here. In the mid-voltage region (70-130 V), the current slope increases due to the tunneling effect. We adjust `jtun` and `ntun` to reproduce this behavior. In the high-voltage region (above 130 V), the current rapidly increases due to the avalanche breakdown. We adjust `bv` and `nbv`, and set `ikr` to the onset of the breakdown (approximately 0.5 μA).

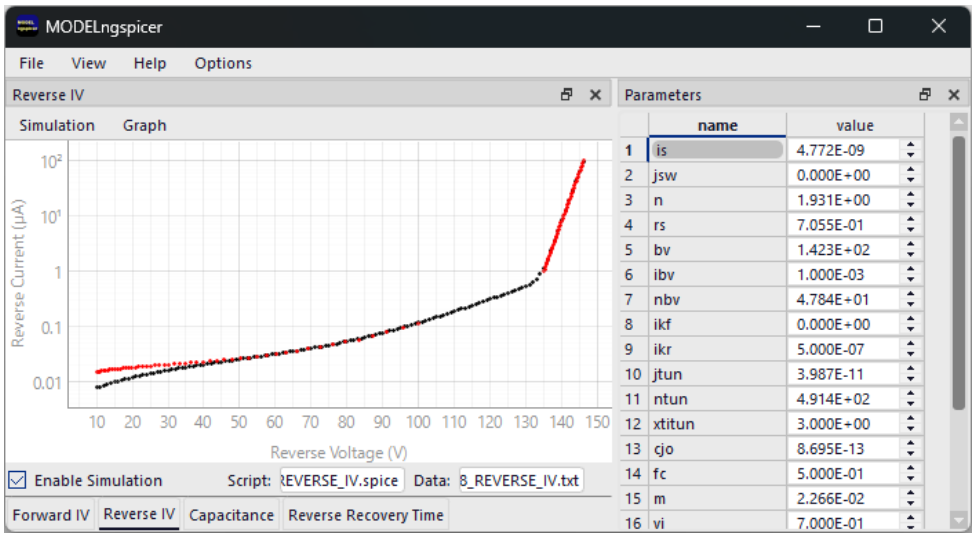

Figure 5: Reverse I-V characteristics

5.2.4 Junction Capacitance

Next, we adjust the model based on the junction capacitance vs. reverse voltage characteristics (C_T-V_R). Listing 12 shows the ngspice script for C_T-V_R.

Listing 12: D_CV.spice

```
D_CV.spice
* This script performs an AC analysis to extract the junction capacitance
* of a diode as a function of reverse bias voltage.

* Diode model
.model DIODE1 D (
.include model.txt
+ )

* Test circuit
VR n01 0 dc 0 ac 1 $ AC voltage source
RS n01 n02 1 $ Sense resistor to measure ac current
D1 0 n02 DIODE1 $ Test diode (reverse biased)

.control
* Reverse voltage sweep parameters
let VR_st = 0 $ Start value of VR
let VR_sp = 15 $ Stop value of VR
let VR_step = 0.1 $ Step size for VR sweep

* Calculate number of steps and initialize index
let count = (VR_sp-VR_st)/VR_step
let index = 0

* Initialize vectors to store CT and VR values
let CT_vector = vector(count)
let VR_vector = VR_st+VR_step*vector(count)

while index lt count
    alter VR dc VR_vector[index] $ Set reverse bias voltage
    ac lin 1 1Meg 1Meg $ AC analysis at 1 MHz

    $ Calculate Y11 and extract capacitance
    let Y11 = v(n01,n02)/v(n02)
    let CT_vector[index] = abs(imag(Y11)/(2*pi*frequency))

    $ Increment loop index
    let index = index + 1
end

* Set scale and write data to file
setscale CT_vector VR_vector
```

```
43 wrdata D_CV.txt CT_vector
44
45 .endc
46 .end
```

Figure 6 shows a screenshot after tuning the C_T-V_R characteristics. Here, we adjust parameters: `cjo` (zero-bias junction capacitance), `m` (area junction grading coefficient), and `vj` (junction potential).

Red dots represent the datasheet readings, while black dots indicate the simulation results. The two sets of data are in good agreement.

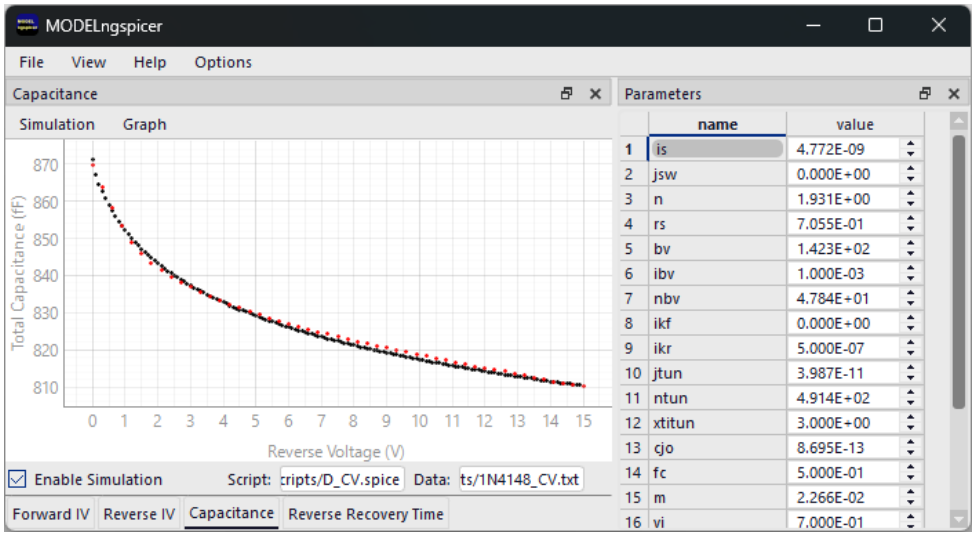

Figure 6: Junction capacitance vs reverse voltage

5.2.5 Reverse Recovery Time

Finally, we adjust the model based on the reverse recovery time (t_{rr}). Reverse recovery time refers to the delay between switching from forward bias to reverse bias, during which excess carriers (electrons and holes) stored in the junction are swept out before the diode fully blocks current.

Figure 7 shows the measurement circuit for reverse recovery characteristics. In this setup, a forward bias current I_F (∼10 mA) is applied via voltage source `VBIAS` and resistor `RBIAS`. Then, a negative pulse is applied from the pulse generator `VPULSE`, and the waveform of the reverse recovery response is observed by a sampling oscilloscope.

Listing 13 shows the ngspice script for t_{rr}.

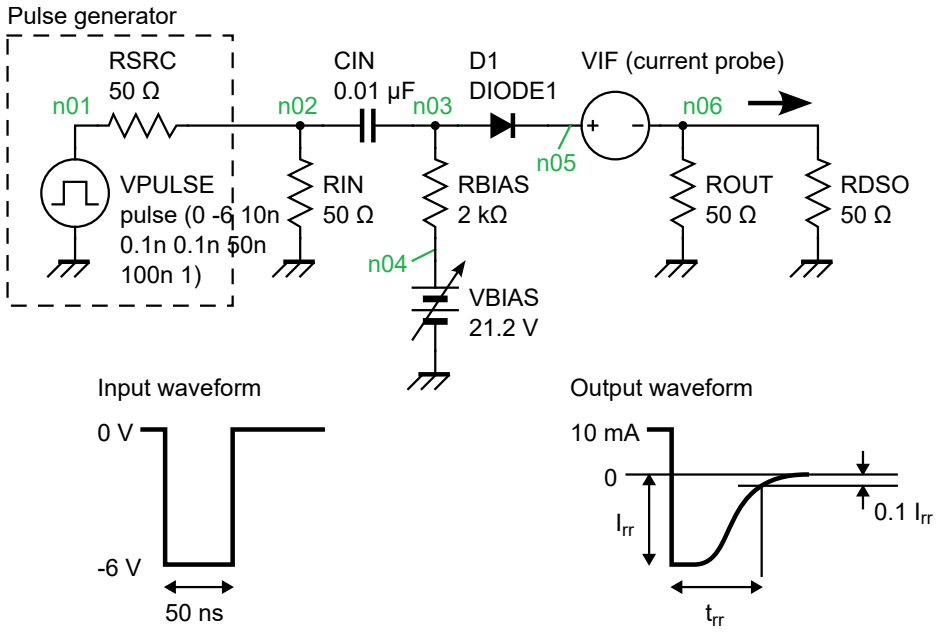

Figure 7: Test circuit for reverse recovery time

Listing 13: D_REVERSE_RECOVERY_TIME.spice

```
D_REVERSE_RECOVERY_TIME.spice
* This script performs a transient analysis to measure the reverse recovery
    time
* of a diode when switching from forward to reverse bias.

* Diode model
.model DIODE1 D (
.include model.txt
+ )

* Test circuit
VPULSE n01 0 dc 0 pulse(0 -6 10n 0.1n 0.1n 50n 100n 1) $ Pulse source
RSRC n01 n02 50 $ Source output resistance (50 Ohm)
RIN n02 0 50 $ Input resistor
```

```
14 CIN n02 n03 0.01u $ DC blocking capacitor
15 RBIAS n03 n04 2k $ Bias resistor to set forward current
16 VBIAS n04 0 dc 21.2 $ Bias voltage source for forward conduction
17 D1 n03 n05 DIODE1 $ Test diode
18 VIF n05 n06 dc 0 $ Current probe for diode
19 ROUT n06 0 50 $ Output resistor
20 RDSO n06 0 50 $ Oscilloscope input resistance (50 Ohm)
21
22 .control
23 * Transient analysis: 10 ps step, 20 ns total time
24 tran 10p 20n
25
26 * Write diode current waveform to file
27 wrdata D_REVERSE_RECOVERY_TIME.txt i(VIF)
28
29 .endc
30 .end
```

Figure 8 shows a screenshot after tuning the t_{rr} characteristics. Here, the parameter `TT` (transit time) is adjusted. According to the datasheet of 1n4148, the reverse recovery time is specified as t_{rr}=3.2 ns under the condition of I_F=10 mA and I_{rr}=10 mA. The simulated waveform successfully reproduces this characteristic.

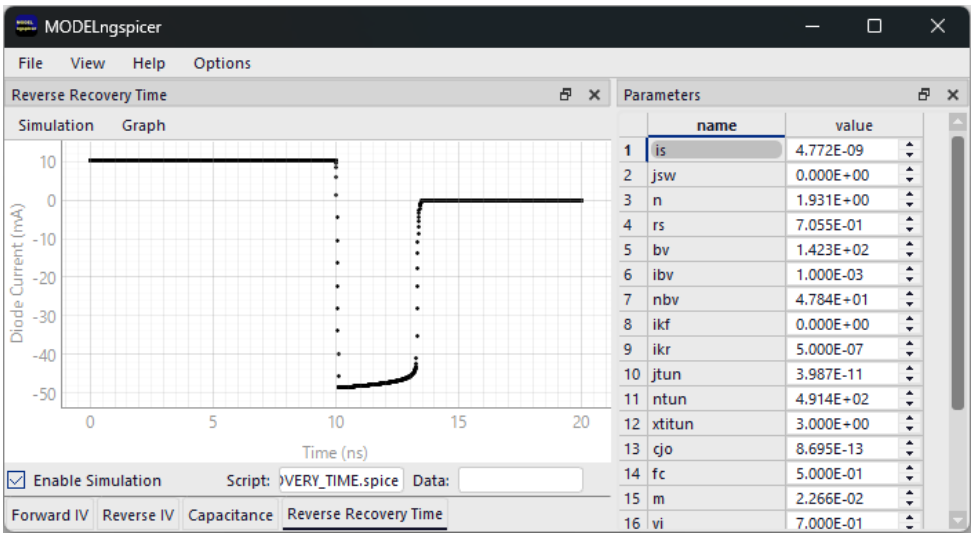

Figure 8: Reverse recovery time