

BULLSEYE Manual

Front-end Design Management

Simshelf International LLC

November 2009

BULLSEYE Manual
Front-end Design Management

BULLSEYE Manual
Front-end Design Management

<i>I.</i>	<i>Design flow in Bullseye</i>	<i>5</i>
<i>II.</i>	<i>Installation</i>	<i>7</i>
1.	Installing the HSPICE simulator	7
2.	Installing Python	7
3.	Installing Bullseye	8
<i>I.</i>	<i>The Bullseye window</i>	<i>10</i>
<i>II.</i>	<i>Preparing the circuit's netlist</i>	<i>11</i>
<i>III.</i>	<i>Defining building block (cell) properties</i>	<i>14</i>
<i>IV.</i>	<i>Optimization parameters (circuit under test)</i>	<i>15</i>
<i>V.</i>	<i>Operating region constraints</i>	<i>16</i>
<i>VI.</i>	<i>Test bench setup</i>	<i>18</i>
1.	Variables	18
2.	Save directives	18
3.	Simulation setup	19
4.	Measurements	22
5.	Model declaration	35
6.	Corner declaration	36
<i>IX.</i>	<i>Optimizer settings</i>	<i>37</i>
<i>X.</i>	<i>Graphic output setup</i>	<i>40</i>
1.	Graph list	40
2.	Trace styles	41
3.	Trace list	42
<i>XI.</i>	<i>Starting a run</i>	<i>43</i>
<i>XII.</i>	<i>Inspecting the results (results browser)</i>	<i>43</i>

BULLSEYE Manual
Front-end Design Management

I. Design flow in Bullseye

Bullseye front-end design management enables you to control your simulations from a simple graphical user interface. After defining the circuit, the design variables, the corners, the simulations, and the measurements you can quickly re-simulate your circuit with changed design variables and rapidly evaluate the impact the change has on the circuit's performance. Adding a corner is simple in Bullseye. Just specify the library and the design parameter values that correspond to the corner and you are ready to re-simulate. Your simulation results can be displayed both numerically and graphically. Bullseye also provides optimization capabilities that can increase your productivity.

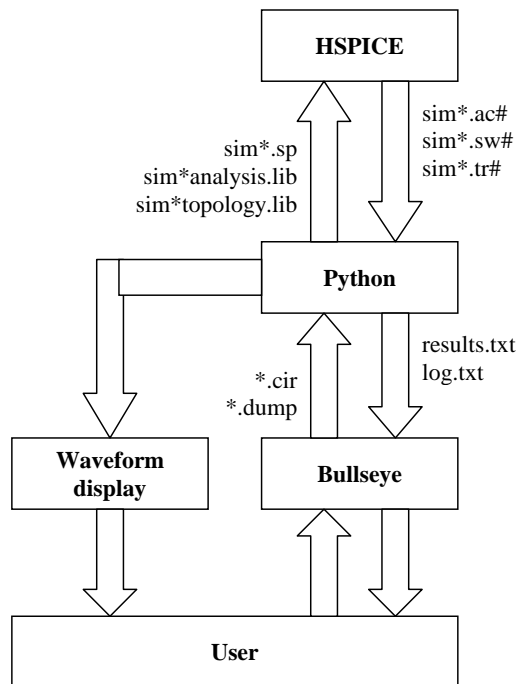


Figure 1: Design flow in Bullseye.

BULLSEYE Manual
Front-end Design Management

II. Installation

1. Installing the HSPICE simulator

You will need a working HSPICE installation. Refer to HSPICE documentation for details. Set the HSPICE_BINARY environmental variable to the full path to the HSPICE simulator executable.

In Windows XP the variable can be set by adding it to “System Variables” under “Control Panel/System/Advanced”. If HSPICE is installed in c:\synopsys\Hspice_Z-2007.03\BIN as hspice.exe the variable should be set to

```
c:/synopsys/Hspice_Z-2007.03/BIN/hspice.exe
```

Note that slashes are used instead of backslashes.

In Linux the variable can be added by editing /etc/profile (if you are using BASH as your shell). Suppose HSPICE is installed in /home/public-software/hspice/linux as hspice. The following two lines must be added to /etc/profile

```
HSPICE_BINARY=/home/public-software/hspice/linux/hspice  
export HSPICE_BINARY
```

Note that if you don't set the HSPICE_BINARY variable the hspice executable must be named hspice under Linux or hspice.exe under Windows and it must be accessible via the system path.

2. Installing Python

For using the Windows version of Bullseye with Python 2.5 the following Python packages must be installed in c:\python25:

```
python-2.5.1.msi  
multiprocessing-2.6.0.2.win32-py2.5.exe  
wxPython2.8-win32-unicode-2.8.9.1-py25.exe  
numpy-1.2.1-win32-superpack-python2.5.exe  
scipy-0.7.0b1-win32-superpack-python2.5.exe  
matplotlib-0.98.3.win32-py2.5.exe
```

The Python binary must be accessible via the system path (add c:\python25 to the PATH system environmental variable).

Unpack cirsimlib.zip to c:\. This results in a directory named c:\cirsimlib. Create a new system environmental variable PYTHONPATH with value set to c:\cirsimlib. If PYTHONPATH variable already exists add a semicolon (;) and c:\cirsimlib to the current value of the PYTHONPATH variable.

For using the Debian Linux version install the following Python packages

```
python
```

BULLSEYE Manual

Front-end Design Management

```
python-wxgtk2.6
python-numpy
python-scipy
python-matplotlib
```

If you are using Python 2.5 you will need the backport of the multiprocessing module for python 2.6. Unpack multiprocessing-2.6.0.2.tar.gz and follow the instructions in INSTALL.txt. If you are using Python 2.6 you don't need to install the backport of the multiprocessing module.

Unpack cirsimlib.tar.gz to whatever directory you like (let's assume you unzipped it to /home/public_software which resulted in a new directory named /home/public_software/cirsimlib). Add the following environmental variable to /etc/profile (if you are using BASH as your shell):

```
PYTHONPATH=/home/public-software/cirsimlib
export PYTHONPATH
```

3. Installing Bullseye

Under Windows unzip bullseye.zip to c:\. This will create the c:\bullseye directory. Add new system environmental variables named BULLSEYEHOME with value set to c:\bullseye. Copy the lock file to c:\bullseye\lib. The Bullseye executable binary is located in c:\bullseye\bin. It is named bullseye.exe. Start it with the -h command line option to invoke the HSPICE mode by typing

```
C:\bullseye\bin\bullseye.exe -h
```

in the command prompt or the "Run..." dialog invoked from the Start menu. It is good practice if you also add c:\bullseye\bin to the system path. This way you can start Bullseye without specifying the full path to it by simply typing

```
bullseye -h
```

in the command prompt or the "Run..." dialog invoked from the Start menu.

Under Linux unzip bullseye.tar.gz to whatever directory you like (let's assume you unzipped it to /home/public_software which resulted in a new directory /home/public_software/bullseye). Copy the lock file to /home/public-software/bullseye/lib/bullseye. Add the following environmental variable to /etc/profile (if you are using BASH as your shell):

```
BULLSEYEHOME=/home/public-software/bullseye
export BULLSEYEHOME
```

To start Bullseye type

```
/home/public-software/bullseye/bin/bullseye -h
```


BULLSEYE Manual

Front-end Design Management

in the shell. You can add /home/public-software/bullseye/bin to the system path. This way Bullseye can be started by typing

```
bullseye -h
```

in the shell.

BULLSEYE Manual

Front-end Design Management

I. The Bullseye window

Bullseye window comprises three panes. The left pane (settings group selector pane) lists the various groups of session settings. The actual settings are entered in the right pane (settings pane). The right pane is divided in two sub panes. Their meaning depends on the settings that are being entered and will be explained in the upcoming sections. The messages resulting from Bullseye's actions are printed in the bottom pane (message pane).

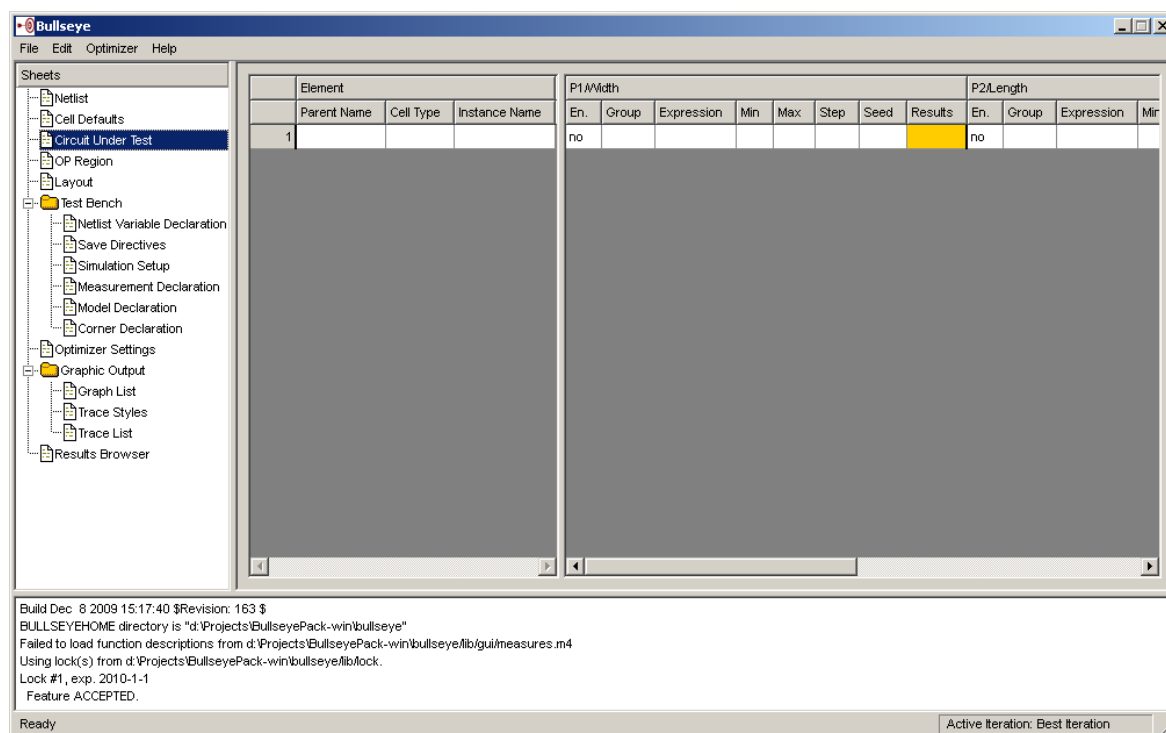


Figure 2: Bullseye window.

Most of the Bullseye's input is in the form of tables. A row is deleted from a table by pressing CTRL+D. A new row is added by pressing CTRL+Enter. A new column can be added or deleted by pressing CTRL+N or CTRL+M, respectively. The contents of a cell can be edited by clicking on the cell and typing. A row can be moved up/down by pressing CTRL+Cursor_up/CTRL+cursor_down.

Rows, columns, and cells can be disabled. A disabled row/column/cell behaves as if it did not exist. A row can be disabled/reenabled by pressing F5. A column's enable state is toggled by pressing F6. A cell can be disabled or enabled by pressing F7. A disabled row/column/cell appears grayed out and cannot be edited.

All these operations can be accessed through using a mouse by right-clicking on a cell and selecting an option from the popup menu.

II. Preparing the circuit's netlist

The netlist syntax adheres mostly to HSPICE rules. The exceptions will be explained in this chapter. The netlist must comprise the definition of the circuit subject to optimization which is also referred to as the circuit under test (CUT). The CUT must be defined as one or more subcircuits (.subckt/.ends) blocks. No libraries that change from corner to corner may be included in the netlist with .include or .lib statements.

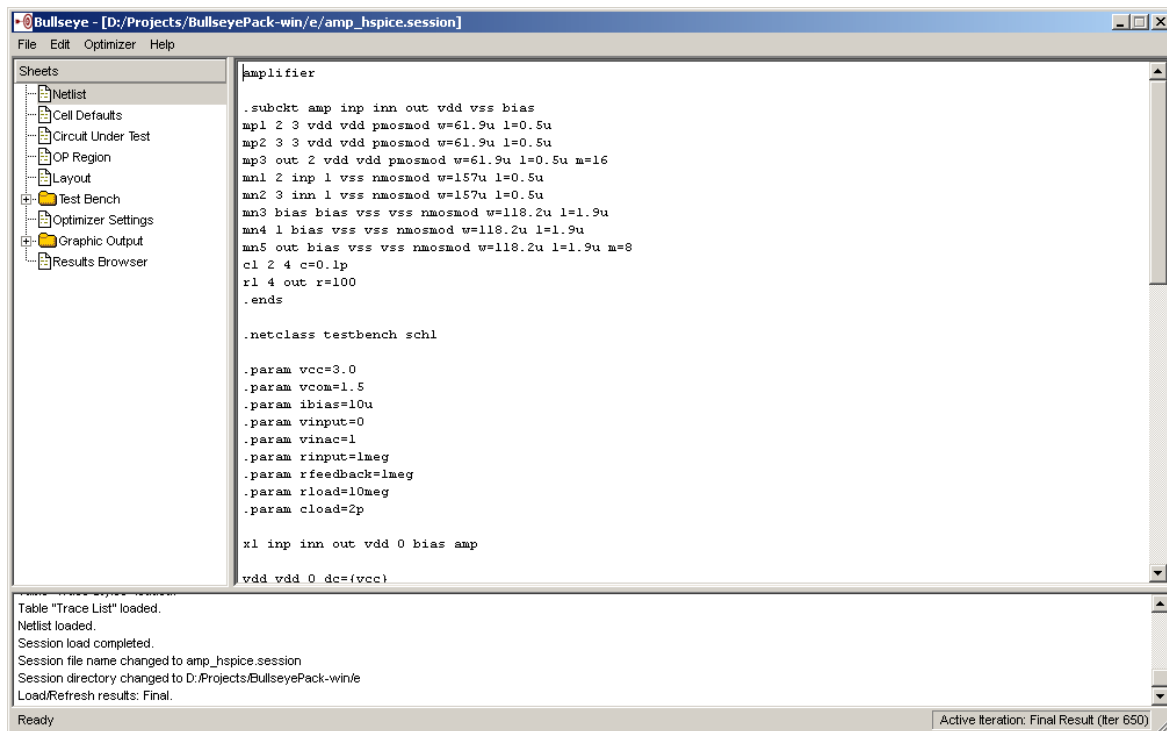


Figure 3: Entering the netlist.

For simulation purposes the CUT must be instantiated in a testbench circuit. Multiple test bench circuits can be defined. To define a test bench circuit you must place it in a .netclass/.endn block using the following syntax:

```
.netclass testbench <topology_name>
<topology description>
.endn
```

Elements in a test bench circuit can be parameterized. All parameters used by elements of a test bench circuit must be defined with .param statements in the .netclass/.endn block. Parameterized expressions used for setting instance parameters must be enclosed in curly braces (e.g. {rin*10}) instead of single quotes (e.g. 'rin*10'). Parameterized expressions on .param lines must not be enclosed in any quotes or braces. Only a single parameter can be defined by one .param line.

The circuit in Figure 4 will be used as the CUT throughout this manual. Figure 5 depicts a test bench circuit used for evaluating the CUT.

BULLSEYE Manual Front-end Design Management

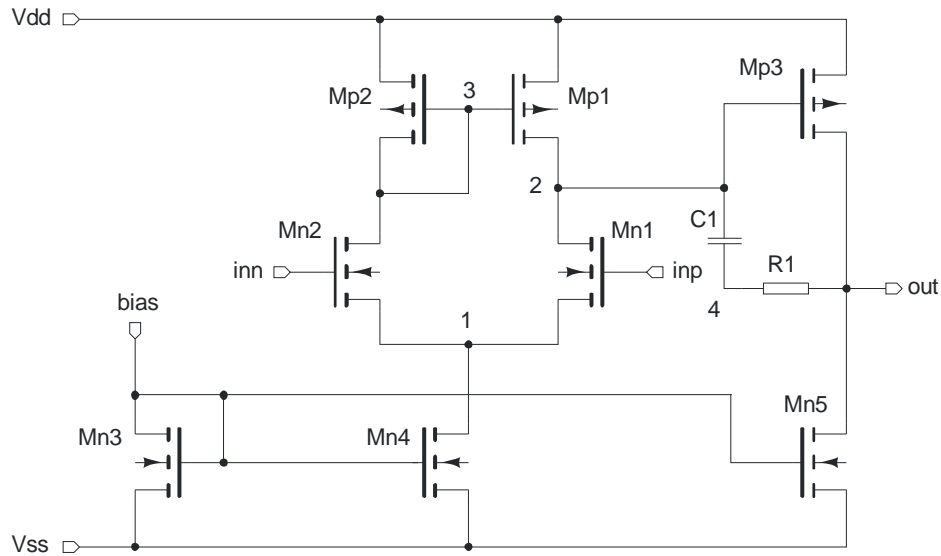


Figure 4: Amplifier schematic. Internal nodes are denoted by numbers. All MOS transistor bulks are connected to Vdd (PMOS) or Vss (NMOS).

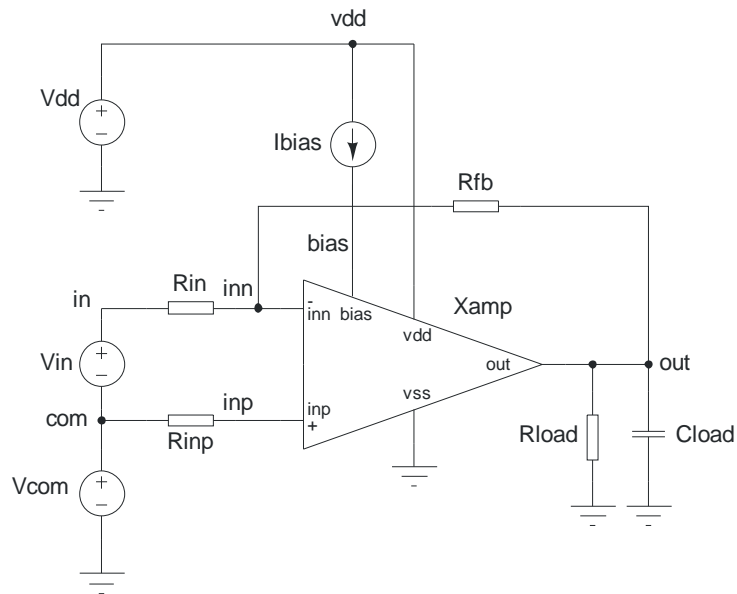


Figure 5: Test bench circuit.

Now let's write down the netlist for the example.

Amplifier

```
.subckt amp inp inn out vdd vss bias
mp1 2 3 vdd vdd pmosmod w=61.9u l=0.5u
mp2 3 3 vdd vdd pmosmod w=61.9u l=0.5u
mp3 out 2 vdd vdd pmosmod w=61.9u l=0.5u m=16
mn1 2 inp 1 vss nmosmod w=157u l=0.5u
mn2 3 inn 1 vss nmosmod w=157u l=0.5u
mn3 bias bias vss vss nmosmod w=118.2u l=1.9u
```

BULLSEYE Manual

Front-end Design Management

```
mn4 1 bias vss vss nmosmod w=118.2u l=1.9u
mn5 out bias vss vss nmosmod w=118.2u l=1.9u m=8
c1 2 4 c=0.1p
r1 4 out r=100
.ends

.netclass testbench sch1

.param vcc=3.0
.param vcom=1.5
.param ibias=10u
.param vinput=0
.param vinac=1
.param rinput=1meg
.param rfeedback=1meg
.param rload=10meg
.param cload=2p

x1 inp inn out vdd 0 bias amp

vdd vdd 0 dc={vcc}
ibias vdd bias dc={ibias}
vcom com 0 dc={vcom}
rinp com inp r=20u
rin in inn r={rinput}
rfb inn out r={rfeedback}
vin in com dc={vinput} ac={vinac}
rload out 0 r={rload}
cload out 0 c={cload}

.endn

.end
```

As you can see the CUT is not parameterized in the netlist. The name of the NMOS/PMOS model is nmosmod/pmosmod. Parameterization is done by Bullseye. Also the library with the MOS models is not included because it changes from corner to corner. There is only one test bench circuit named sch1. The test bench circuit is parameterized and the parameters are defined in the .netclass/.endn block. When the problem is exported and run Bullseye parameterizes the CUT, processes .netclass/.endn blocks, and inserts .lib statements for the inclusion of MOS models.

III. Defining building block (cell) properties

Every CUT consists of building blocks (cells). Basic cell properties required by the optimization process are entered in the Cell Defaults settings group. Every cell has a name. In our example we have four types of cells: nmos (cell name nmosmod), pmos (cell name pmosmod), resistor (cell name res), and capacitor (cell name cap).

The elt. Inst. field is required only if a cell is defined as a subcircuit in which case it specifies the name of the instance in the subcircuit that represents the actual device (e.g. MOS transistor). This field is required only if you intend to put any operating point requirements on MOS transistor cells that are defined as subcircuits.

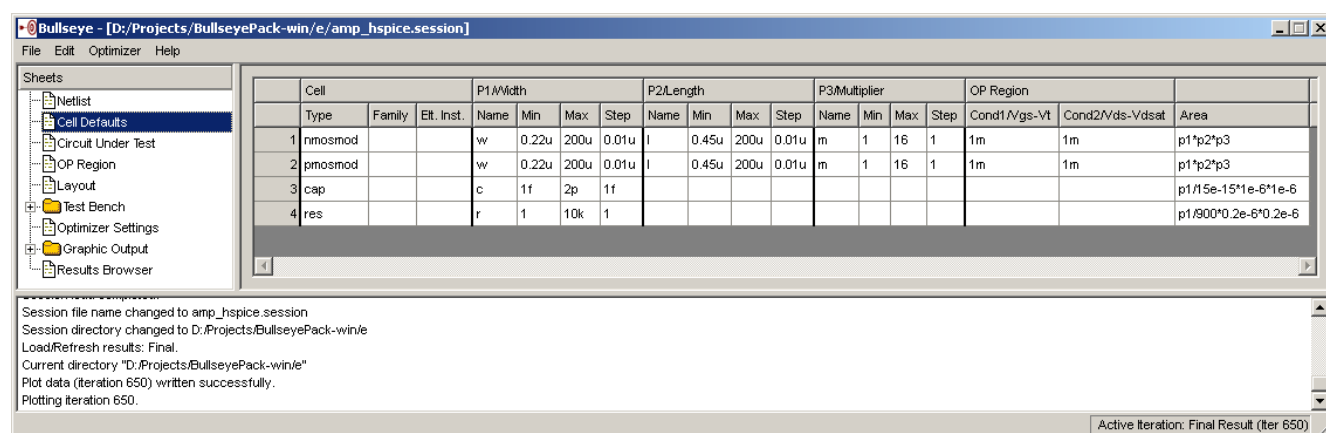


Figure 6: Cell defaults.

Every cell can have up to 3 parameters that can be subject to optimization. Typically for MOS transistors these are channel width, channel length, and multiplier. If a cell has less than 3 parameters describe only those that are available and leave the rest of the cell defaults empty.

For every parameter its name must be specified. For the sake of optimization the minimal and the maximal allowed value of a parameter must also be specified. Step specifies the size of increment used for modifying the parameter. For MOS cells you can also specify the default operating point requirements for minimal Vgs-Vt and Vds-Vdsat at the operating point of the circuit.

Every cell contributes to circuit area. The area contribution is calculated according to the formula specified in the Area column. The formula may not use Si prefixes for numeric values (e.g. use 1e-6 instead of 1u). Parameters of a cell can be entered into the formula by using p1, p2, and p3.

The cell defaults for the example circuit are depicted by Figure 6.

IV. Optimization parameters (circuit under test)

The cell instances that are subject to optimization must be specified in the Circuit Under Test settings group. Every cell instance has a parent name (e.g. the name of the subcircuit definition where the cell instance is defined). In our example this name is amp for all cell instances.

The cell type is the name of the cell type. This name links a row in the CUT definition table with a corresponding line in the Cell Defaults table. Finally every cell instance has an instance name which is the name used for instantiating the cell in the netlist.

For every one of the 3 available optimization parameters one can specify overrides for the minimal value, maximal value, and the step used by the optimizer. If these fields are left empty the defaults from the Cell Default table are used. Seed specifies the initial value of the parameter. The result column displays the result of the active iteration (the one that is highlighted in the Results Browser). The results relative position to the minimal and maximal value is indicated by a bar in the result cell.

The En. Column enables or disables the optimization of a parameter (yes stands for enable). If a parameter is disabled, its value is equal to the value specified in the Seed column and is untouched by the optimizer. Note that the values of the parameters specified in the netlist are ignored.

The group column specifies the name of the group to which this parameter belongs. Parameters in the same group have identical values and are modified by the optimizer in a parallel way (one can also say that these parameters are matched). Finally the Expression column enables you to specify an expression for a parameter. An expression allows for more flexible ways of matching one parameter with another. Expressions can be of the form $a*x+b$ where a and b are constants and x is an identifier. Expressions override the Group setting.

BULLSEYE Manual

Front-end Design Management

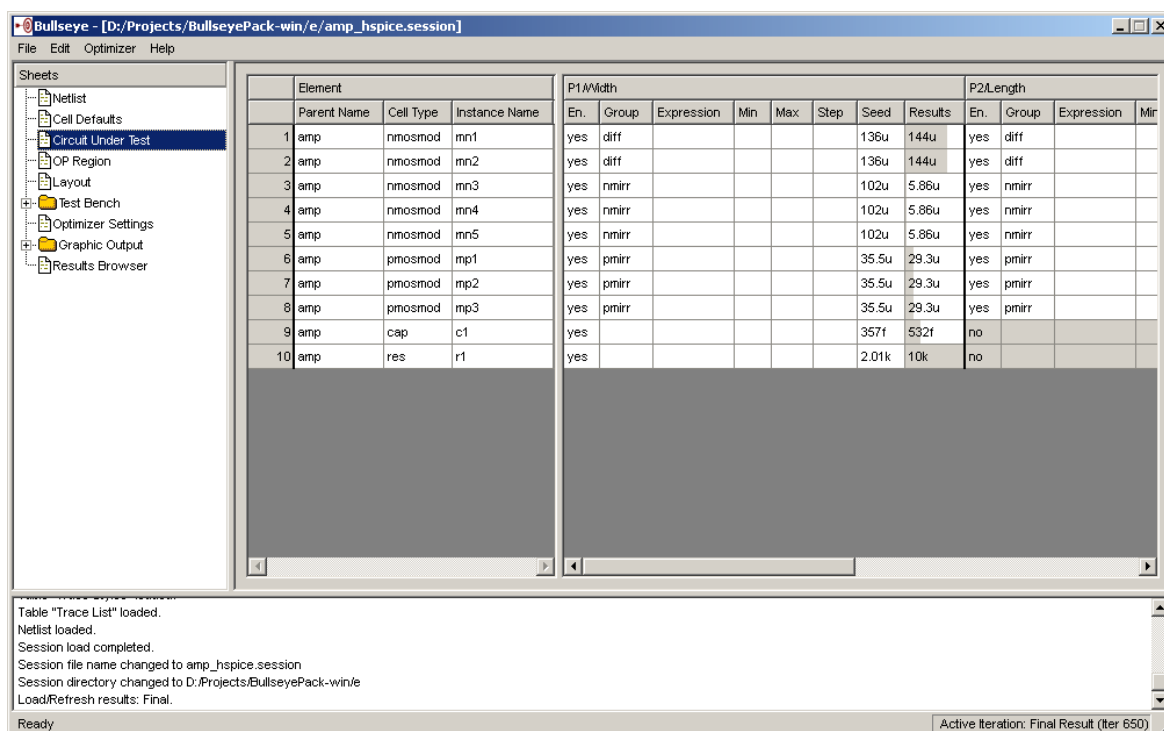


Figure 7: Defining the circuit under test (CUT).

The En. Through Results columns are repeated for the remaining two parameters of every cell. If a cell has less than 3 parameters some columns may be disabled.

The Group setting is valid for matching only between the same parameters of the same column (e.g. width of one MOS to the width of another MOS). The Expression makes it possible to match between columns (e.g. you can match width of one MOS to the length of the same MOS or even the length of a different MOS). The identifier in an expression has nothing to do with the name of a group. You can have the same name for a first parameter group and for an identifier (say A) without a resulting matching between the first parameter of cells that belong to first parameter group A and the cells with an expression containing identifier A.

V. Operating region constraints

You can specify constraints on the operating point of the circuit, specifically on the differences $V_{gs}-V_t$ and $V_{ds}-V_{dsat}$. The column Parent Instance specifies the name of the instance in the test bench circuit where the specific MOS resides that is the subject of a constraint on $V_{gs}-V_t$ and $V_{ds}-V_{dsat}$. Cell type specifies the name of the corresponding definition in the Cell Defaults table. Instance Name specifies the name of the actual cell instance on which the constraint is enforced.

BULLSEYE Manual

Front-end Design Management

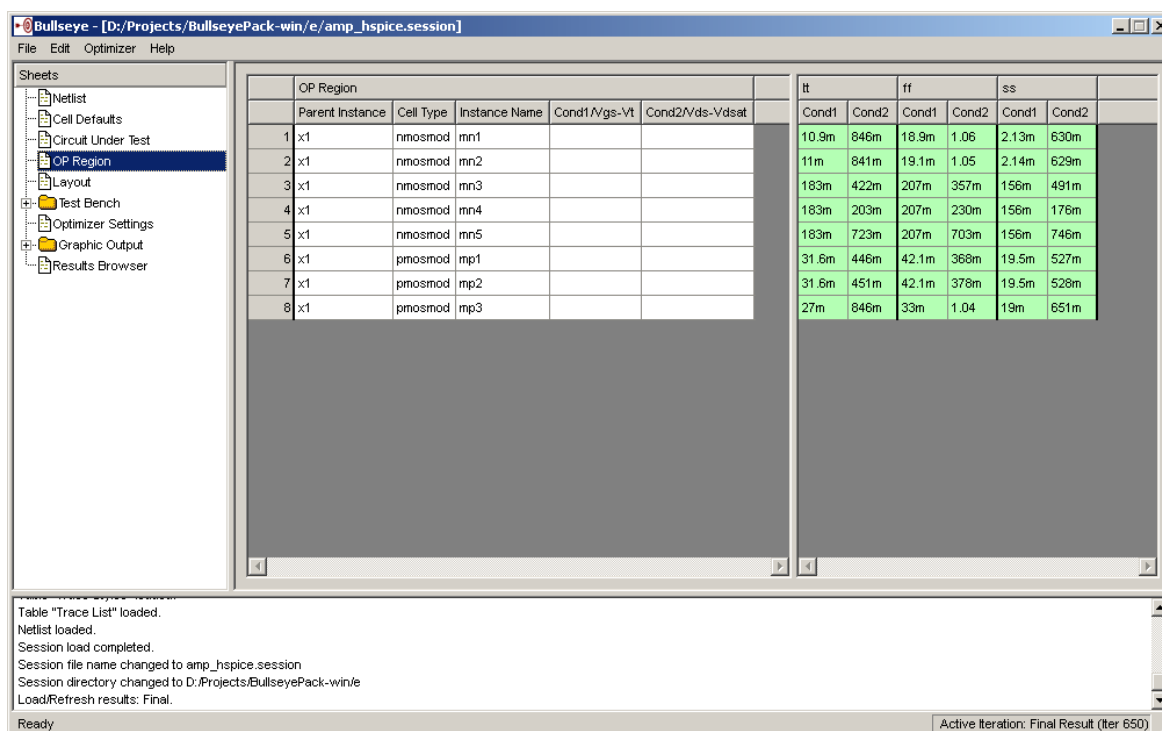


Figure 8: Operating region constraints.

Columns Cond1 and Cond2 specify the minimal value of $V_{gs}-V_t$ and $V_{ds}-V_{dsat}$. If no value is specified for Cond1 or Cond2 the value is taken from the Cell Defaults table. Any of the Cond1 and Cond2 cells can be disabled (F8). If a cell is disabled the corresponding operating point constraint is not enforced. A disabled cell can be enabled by pressing F8.

The operating point constraints are enforced on the operating point results obtained by the analysis with the name "op" (see section Simulation setup VI.3).

The right sub pane of the settings pane lists the value of $V_{gs}-V_t$ and $V_{ds}-V_{dsat}$ for all corners of the active iteration (the one that is highlighted in the Results Browser). Green fields denote an operating condition that satisfies the requirements while red fields denote a failure to satisfy the requirements.

VI. Test bench setup

1. Variables

The Netlist Variable Declaration settings group enables you to specify the values of the variables (the ones defined with .param statements in test bench circuit definitions). All variables defined in all test bench circuit definitions are listed in the table. By clicking on the cells of the Depends On column you can select whether a variable will behave as a constant (constant), change its value depending on the corner (corner), or change its value depending on the analysis (analysis) that is being performed.

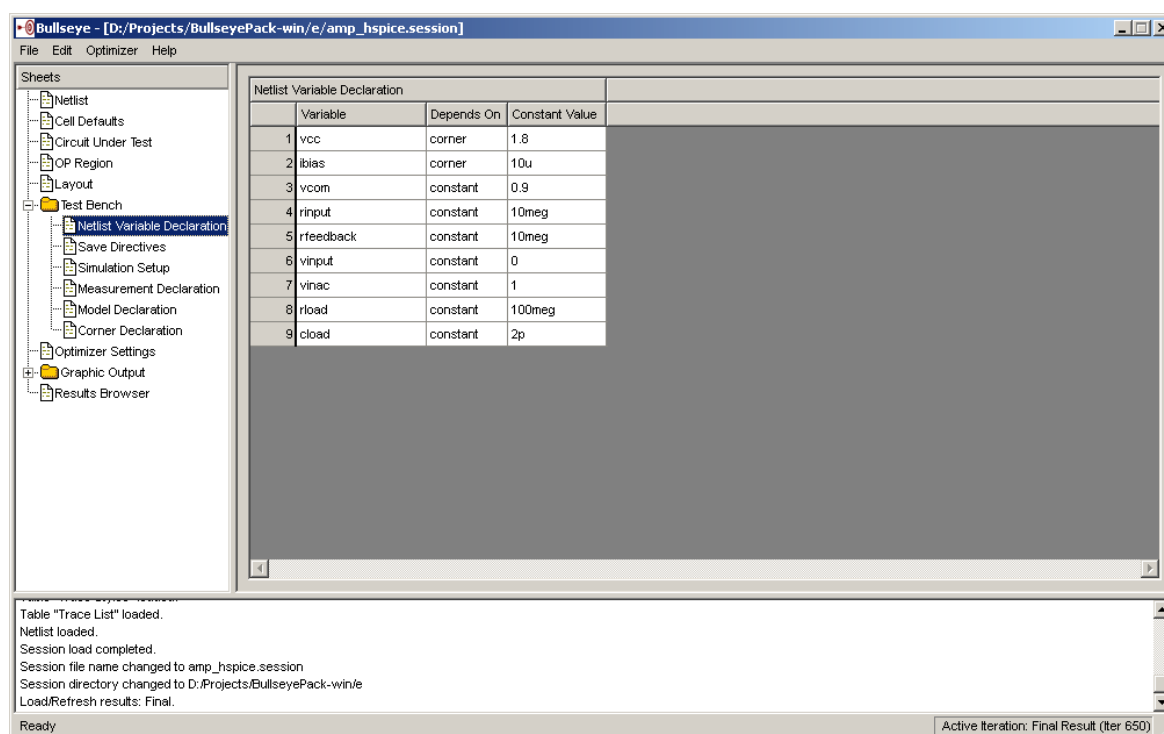


Figure 9: Defining variables.

For constant variables the value can be specified in the Constant Value column. All variables must have a value defined on the Constant Value column. The value specified in the .param statement is ignored by and the one specified in Netlist Variable Declaration is used.

The values of variables that depend on the corner or on the analysis are specified in the Simulation Setup (see section VI.3) or in Corner Declaration (see section VI.6).

2. Save directives

The simulator by default saves node voltages and currents of voltage sources as simulation results. If you want to narrow down the number of saves vectors (and by that speed up the process of optimization) you can specify what results you want to save during the analysis. In the Save Directives table every column specifies one group of save directives. Columns can be added by pressing CTRL+N and deleted by

BULLSEYE Manual

Front-end Design Management

pressing CTRL+M. The first row in a column specifies the name of the group of save directives. The remaining rows specify the save directives, one per line.

Later in the Simulation Setup settings group you can specify one or more groups of save directives for every analysis.

Save directives can be:

- a node voltage – specified as **v(['node_name'])**
- instance current – specified as **i(['device_name'])**
- instance property – specified as **p(['instance_name'], ['property_name'])**

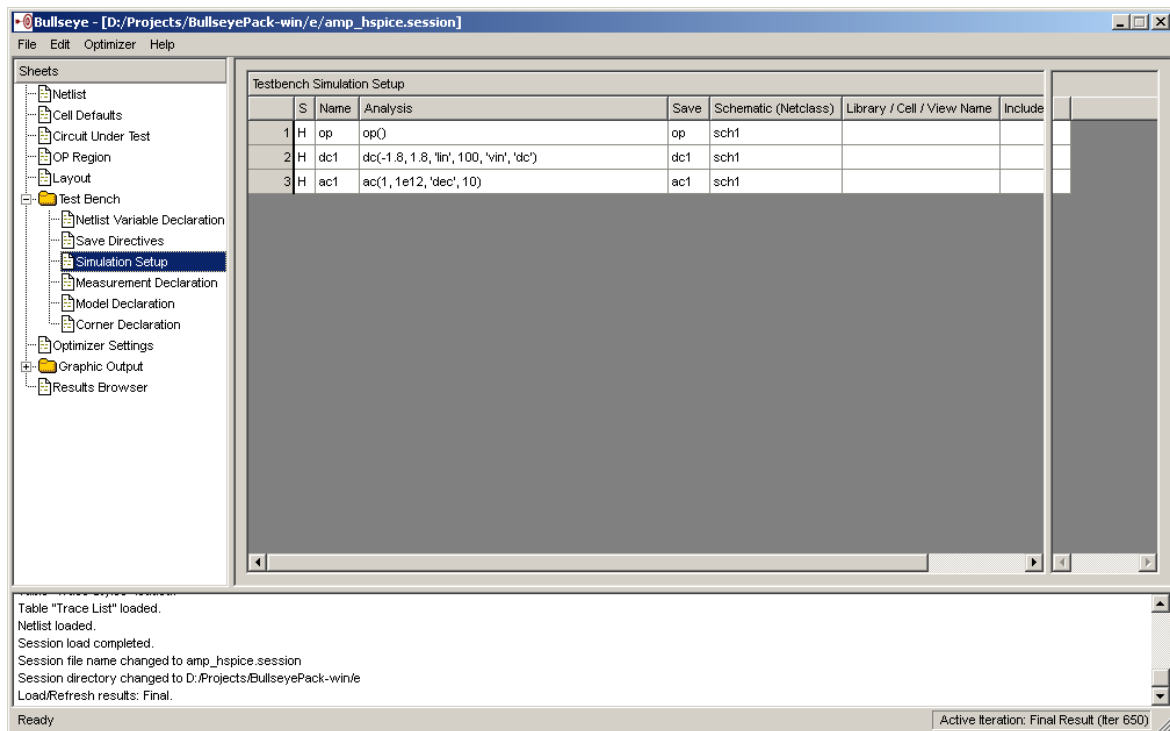


Figure 10: Save directives.

Empty cells in Save Directives produce no save directives for the simulator. Save directives are ignored in AC analysis because of the way they are processed by HSPICE (using the .probe statement). In AC analysis all node voltages and all voltage source currents are always saved, regardless of save directives.

3. Simulation setup

In the simulation setup you can define the analyses that will be performed on your test bench circuits.

BULLSEYE Manual

Front-end Design Management

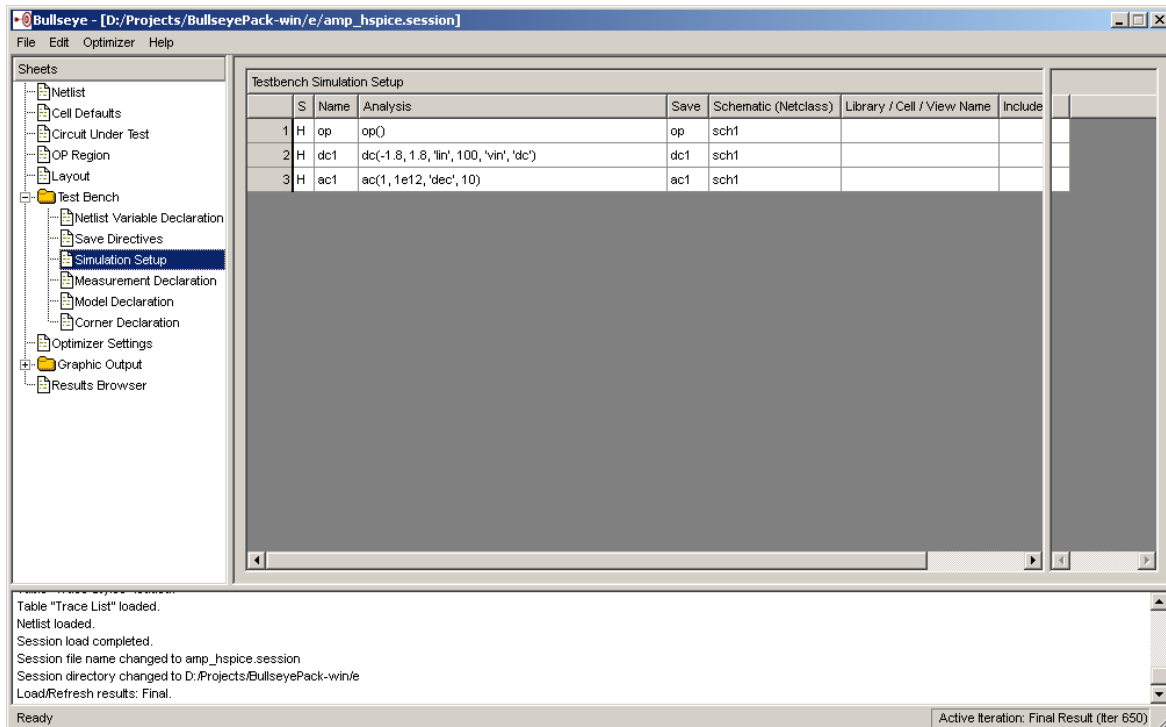


Figure 11: Simulation setup.

Column Name specifies the name of the analysis. Save specifies a space separated list of save directive group names (see section VI.2). The topology that is used for the analysis (the test bench circuit) is specified by listing its name in the Schematic (Netclass) column. Analyses are defined in the Analysis column using one of the following statements.

- **op()** – operating point analysis
- **dc(start, stop, step_type, nsteps, instance, parameter)** – operating point sweep
- **ac(fstart, fstop, step_type, npoints)** – small signal AC analysis
- **tran(tstep, tstop, tstart, maxStep, uic_flag)** – transient analysis
- **noise(fstart, fstop, step_type, npoints, input_instance, positive_output, negative_output)** – noise analysis

In dc() analysis start and stop specify the initial and the final point of the sweep while step_type can be 'lin' (linear sweep with npoints points), 'dec' (logarithmic sweep with npoints points per decade), or 'oct' (logarithmic sweep with npoints points per octave). Instance and parameter define the name of the instance and its parameter that is swept. To sweep the temperature set the last two parameters to None and temperature, respectively.

In ac() analysis the range of the frequency sweep is defined with fstart and fstop while step_type and nsteps have the same meaning as in dc() analysis.

In tran() analysis the first two parameters (tstep and tstop) specify the initial step and the duration of the analysis. Both of them are mandatory while the others are optional. Parameter tstart specifies the starting time at which the simulator starts recording results (default is 0). maxStep (if specified) sets an upper bound on the time step. If uic_flag is set to True, the simulator starts the simulation from a point defined

BULLSEYE Manual

Front-end Design Management

by the initial conditions specified using .ic statements. Otherwise the simulation is started from the operating point results.

Parameters of noise() analysis have the same meaning as for the ac() analysis. Additionally input_instance specifies the input independent voltage or current source where the equivalent input noise is calculated, while positive_output and negative_output specify the names of the positive and the negative node of the output where the noise is measured. The result of the noise() analysis are noise power density spectra.

Analysis with the name 'op' is special. This analysis must be an op() analysis and represents the source of $V_{gs}-V_t$ and $V_{ds}-V_{dsat}$ values for enforcing the operating point constraints defined in the OP Region table.

Variables marked as analysis dependent in the Variable Declaration settings group appear in the right sub pane of the settings pane. The values of all such variables must be defined for all of the analyses even if the value is not actually used in the test bench circuit.

BULLSEYE Manual

Front-end Design Management

4. Measurements

Bullseye performs measurements on simulation results by evaluating NumPy expressions. The Analysis column lists the name of the analysis (corresponds to the Name column in Simulation Setup table) that produces the results used as input to the measurement. Corner specifies the space separated list of corners in which the measurement is performed. Asterisk (*) stands for all defined corners. Name specifies the name of the measurement.

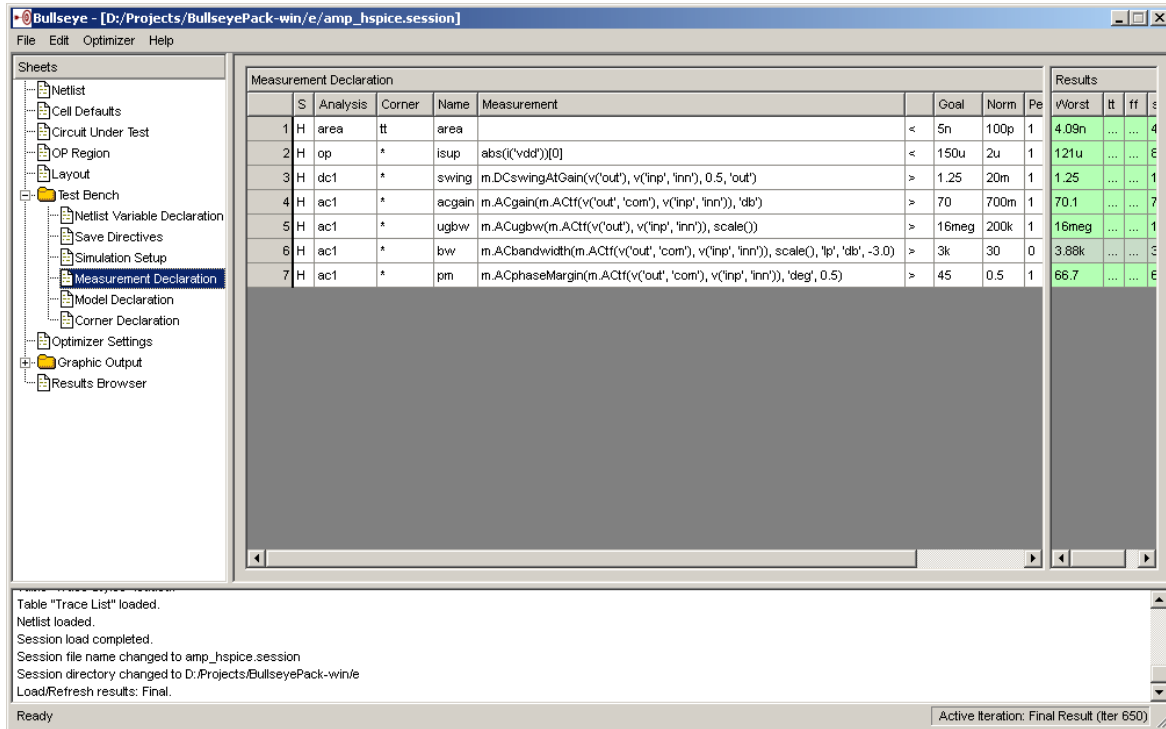


Figure 12: Defining the measurements.

The NumPy expression that produces the value of the measurement is listed in column Measurement. Setting the Analysis column to area results in the area measurement. For the area measurement the Measurement should be left blank. The value of the area measurement is the sum of area expression values for all cells specified in the Circuit Under Test table. The area expressions for individual cell types are listed in the Cell Defaults table in the Area column (see section III).

The simulation results and the variables are accessed using the following functions/expressions.

- **v('name')** or **v('namep', 'namen')**
The first form returns the node voltage at node 'name' while the second one returns the voltage between nodes 'namep' and 'namen'.
- **i('name')**
Returns the current flowing through voltage source with named 'name'.
- **p('instance_name', 'property_name')**
Returns the value of property 'property_name' of instance 'instance_name'. Note that the desired property must be listed in the Save Directives table and the name of the corresponding save

BULLSEYE Manual

Front-end Design Management

directive group must be specified in the Save column of analysis that produces the results that are being processed.

- **scale()**
Returns the default scale of the analysis results.
- **param['name']**
Returns the value of variable named 'name'.

If the analysis name is not specified in the Analysis column the measurement is performed after all measurements that have an analysis specified have been evaluated. Such measurement can't access voltages, currents, or instance properties calculated by the simulator because there is no analysis specified that could provide them. Instead they can use the results of other measurements using the following syntax.

result['measure_name']['corner_name']

For such measurements **thisCorner** results in the name of the current corner in which the measurement is being evaluated.

All of the NumPy functions are available for building expressions (as if 'from numpy import *' had been specified in the source file). Additional functions enable signal manipulations and measurements common in electrical and electronics engineering. These functions can be found in Python module **m** and are accessed as **m.measurement_name**. The following specialized measurements are available (default values are specified in Python syntax).

Deg2Rad(x)

Input

x .. NumPy array of real values representing angles in degrees

Output

NumPy array of angles in radians

Converts degrees to radians.

Rad2Deg(x)

Input

x .. NumPy array of real values representing angles in radians

Output

NumPy array of angles in degrees

Converts radians to degrees.

dB2gain(x, unit='db20')

Input

x NumPy array of real values representing gain values
unit string specifying the unit for gain
 Possible values of unit are 'db', 'db20', and 'db10'.

Output

NumPy array of gain magnitudes.

BULLSEYE Manual
Front-end Design Management

Converts gain in decibels of gain (db or db20) or decibels of power gain (db10) to gain magnitude.

gain2dB(x, unit='db20')

Input

x	NumPy array of real values representing gain magnitudes
unit	string specifying the unit for gain
	Possible values of unit are 'db', 'db20', and 'db10'.

Output

NumPy array of gain magnitudes.

Converts gain magnitude to decibels of magnitude (db or db20) or decibels of power (db10).

BULLSEYE Manual
Front-end Design Management

XatI(x, i)

Input

x NumPy 1-dimensional array of values.
i fractional index ranging from 0 to len(x)-1
 If it is a vector the result is also a vector.

Output

The value/values of x at fractional index/indices i obtained through linear interpolation.

Performs linear interpolation based on fractional index.

IatXval(x, val, slope='any')

Input

x NumPy 1-dimensional array of real values
val scalar real value
slope type of slope where intersection of x and val is taken into account
 Can be 'rising', 'falling', or 'any'.

Output

1-dimensional numpy array of fractional indices corresponding to intersections between x and val.

Find intersections of a vector and a scalar (table lookup). Result is in the form of fractional indices.

filterI(i, direction='right', start=None, includeStart=False)

Input

i 1-dimensional NumPy array of fractional indices (possibly obtained from IatXval)
direction direction of search for valid indices
 Can be 'left' or 'right'.
start Initial fractional index.
 Default is i[0] for direction='right', and i[-1] for direction='left'.
includeStart Include fractional indices that are equal to start in result.

Output

For direction 'right' ('left') returns fractional indices that are greater or equal than (less or equal than) start. If includeStart is False the comparison operators are greater than (less than).

Filters fractional indices.

BULLSEYE Manual
Front-end Design Management

XatIrange(x, i1, i2=None)

Input

x 1-dimensional NumPy real array
i1 initial fractional index
i2 final fractional index
 Defaults to len(x)-1.

Output

Return a subvector of x corresponding to endpoint indices i1 and i2. Uses linear interpolation at endpoints.

Return a subvector with fractional indexing and linear interpolation.

dYdI(y)

Input

y 1-dimensional NumPy real array

Output

NumPy array representing the derivative of y with respect to vector index.

Calculate derivative with respect to vector index.

dYdX(y,x)

Input

y 1-dimensional NumPy real array
x 1-dimensional NumPy real array

Output

NumPy array representing the derivative of y with respect to x. Array lengths must match.

Calculate derivative with respect to vector x.

integYdX(y, x)

Input

y 1-dimensional NumPy real array
x 1-dimensional NumPy real array

Output

NumPy array representing the definite integral of y from x[0] to x[i]. Array lengths must match.

Calculate definite integral with respect to vector x.

BULLSEYE Manual
Front-end Design Management

DCgain(output, input)

Input

output 1-dimensional NumPy real array of output values

input 1-dimensional NumPy real array of input values

Output

NumPy array representing the derivative of output with respect to input.

Calculate differential DC gain.

DCswingAtGain(output, input, relLevel, type='out')

Input

output 1-dimensional NumPy real array of output values

input 1-dimensional NumPy real array of input values

relLevel relative gain level with respect to maximal gain considered as swing boundary

type output value type

Can be 'out' (return swing at output) or 'in' (return swing at input).

Output

Output or input swing range corresponding to [relLevel*Amax, Amax] differential gain range.

Calculate DC input or output swing for given differential gain range.

ACcircle(unit='deg')

Input

unit unit for output value

Can be 'deg' (degrees) or 'rad' (radians).

Output

Size of full circle (360 degrees or 2*pi radians).

Return the angle corresponding to full circle.

ACtff(output, input)

Input

output 1-dimensional NumPy real or complex array of AC output values

input 1-dimensional NumPy real or complex array of AC input values

Output

Complex transfer function from input to output.

Calculate complex transfer function from complex AC results.

BULLSEYE Manual
Front-end Design Management

ACmag(tf, unit='db')

Input

tf	1-dimensional NumPy complex array representing the transfer function
unit	unit for output Can be 'db', 'db20' (decibels of gain), db10 (decibels of power), or 'abs' (absolute magnitude).

Output

Magnitude of complex transfer function.

Calculate the magnitude of complex transfer function.

ACphase(tf, unit='deg', unwrapTol=0.5)

Input

tf	1-dimensional NumPy complex array representing the transfer function
unit	unit for output Can be 'deg' (degrees) or 'rad' (radians).
unwrapTol	angle unwrap tolerance relative to pi

Output

Phase of complex transfer function.

Calculate the phase of complex transfer function.

ACgain(tf, unit='db')

Input

tf	1-dimensional NumPy complex array representing the transfer function
unit	unit for output value

Output

Maximal gain magnitude of complex transfer function.

Calculate the maximal gain magnitude of complex transfer function.

ACbandwidth(tf, scl, filter='lp', unit='db', level=-3.0)

Input

tf	1-dimensional NumPy complex array representing the transfer function
scl	1-dimensional NumPy complex array representing the frequency scale
filter	filter type for bandwidth calculation Can be 'lp' (low-pass), 'hp' (high-pass), or 'bp' (band-pass).
unit	unit for the level parameter Can be 'db', 'db20', 'db10', or 'abs'.
level	level defining the edge of pass-band If unit='abs' the level is a multiplier for maximal absolute gain magnitude.

Output

Bandwidth of a complex transfer function.

Calculate the bandwidth of a complex transfer function.

BULLSEYE Manual
Front-end Design Management

ACugbw(tf, scl)

Input

tf 1-dimensional NumPy complex array representing the transfer function
scl 1-dimensional NumPy complex array representing the frequency scale

Output

Unity gain bandwidth (frequency where absolute gain magnitude reaches 1).

Calculate the unity gain bandwidth of a complex transfer function.

ACphaseMargin(tf, unit='deg', unwrapTol=0.5)

Input

tf 1-dimensional NumPy complex array representing the transfer function
unit unit for the output value
 Can be 'deg' (degrees) or 'rad' (radians).
unwrapTol angle unwrap tolerance relative to pi

Output

Phase margin of a complex transfer function (difference between the phase when absolute gain magnitude reaches 1 and -180 degrees).

Calculate the phase margin for a complex transfer function.

ACgainMargin(tf, unit='db', unwrapTol=0.5)

Input

tf 1-dimensional NumPy complex array representing the transfer function
unit unit for the output value
 Can be 'db', 'db20', 'db10', or 'abs'.
unwrapTol angle unwrap tolerance relative to pi

Output

Gain margin of a complex transfer function (how much the gain must increase from the level corresponding to -180 degrees phase to reach unity gain).

Calculate the gain margin for a complex transfer function.

BULLSEYE Manual
Front-end Design Management

Tdelay(

sig1, sig2, scl,
lev1type='rel', lev1=0.5, edge1='any', skip1=0,
lev2type='rel', lev2=0.5, edge2='any', skip2=0,
t1=None, t2=None

)

Input

sig1	1-dimensional NumPy real array representing the first signal
sig2	1-dimensional NumPy real array representing the second signal
scl	scale common to both signals
lev1type	type of level for finding the reference point in the first signal Can be 'abs' (absolute signal value) or 'rel' (relative signal value).
lev1	level defining the reference point in the first signal
edge1	type of edges to consider when searching for crossings between sig1 and lev1 Can be 'rising', 'falling', or 'any'. See latXval().
skip1	number of crossings in sig1 to skip before the reference point is reached
lev2type	same as lev1type, except that it applies to sig2
lev2	same as lev1, except that it applies to sig2
edge2	same as edge1, except that it applies to sig2
skip2	same as skip1, except that it applies to sig2
t1	starting point of scale range that is considered Default is beginning of scale.
t2	end point of scale range that is considered Default is end of scale.

Output

Difference in scale between the reference points in sig2 and sig1.

Calculates the delay between reference points in two signals. If lev1type or lev2type is 'rel' the absolute signal value is calculated relative to the levels of sig1 and sig2 at t1 and t2.

Tovershoot(sig, scl, t1=None, t2=None, outputType='rel')

Input

sig	1-dimensional NumPy real array representing the signal
scl	1-dimensional NumPy real array representing the scale for the signal
t1	starting point of scale range that is considered Default is beginning of scale.
t2	end point of scale range that is considered Default is end of scale.
outputType	type of output value Can be 'rel' (relative to difference between signal values at t1 and t2) or 'abs' (actual overshoot value).

Output

Overshoot of a signal.

Calculates the overshoot of a signal.

BULLSEYE Manual
Front-end Design Management

Tundershoot(sig, scl, t1=None, t2=None, outputType='rel')

Input

sig	1-dimensional NumPy real array representing the signal
scl	1-dimensional NumPy real array representing the scale for the signal
t1	starting point of scale range that is considered Default is beginning of scale.
t2	end point of scale range that is considered Default is end of scale.
outputType	type of output value Can be 'rel' (relative to difference between signal values at t1 and t2) or 'abs' (actual overshoot value).

Output

Undershoot of a signal.

Calculates the undershoot of a signal.

**TedgeTime(
edgeType, sig, scl,
lev1type='rel', lev1=0.1,
lev2type='rel', lev2=0.9,
t1=None, t2=None**

)

Input

edgeType	type of edge to consider Can be 'rising', 'falling', or 'any'. See IatXval().
sig	1-dimensional NumPy real array representing the signal
scl	1-dimensional NumPy real array representing the scale for the signal
lev1type	type of level for finding the first reference point Can be 'abs' (absolute signal value) or 'rel' (relative signal value).
lev1	level defining the first reference point
lev2type	type of level for finding the second reference point Can be 'abs' (absolute signal value) or 'rel' (relative signal value).
lev2	level defining the second reference point
t1	starting point of scale range that is considered Default is beginning of scale.
t2	end point of scale range that is considered Default is end of scale.

Output

Rise/fall time of a signal.

Calculates the edge time (rise/fall) between two reference points in a signal. If lev1type or lev2type is 'rel' the absolute signal value is calculated relative to the levels of the signal at t1 and t2.

```
TriseTime(
    sig, scl,
    lev1type='rel', lev1=0.1,
    lev2type='rel', lev2=0.9,
    t1=None, t2=None
)
```

Same as TedgeTime with edgeType='rising'.

```
TfallTime(
    sig, scl,
    lev1type='rel', lev1=0.1,
    lev2type='rel', lev2=0.9,
    t1=None, t2=None
)
```

Same as TedgeTime with edgeType='falling'.

```
TslewRate(
    edgeType, sig, scl,
    lev1type='rel', lev1=0.1,
    lev2type='rel', lev2=0.9,
    t1=None, t2=None
)
```

The difference between signal at t1 and t2 divided by the result of TedgeTime().

TsettlingTime(sig, scl, tolType='rel', tol=0.05, t1=None, t2=None)

Input

sig	1-dimensional NumPy real array representing the signal
scl	1-dimensional NumPy real array representing the scale for the signal
tolType	type of settling tolerance Can be 'abs' (absolute signal value) or 'rel' (relative signal value).
tol	settling tolerance
t1	starting point of scale range that is considered Default is beginning of scale.
t2	end point of scale range that is considered Default is end of scale.

Output

Settling time of a signal.

Calculates the time from t1 to the moment when the signal settles within settling tolerance of value at t2. If tolType is 'rel' the settling tolerance is calculated by multiplying tol with the difference of signal values at t1 and t2.

BULLSEYE Manual Front-end Design Management

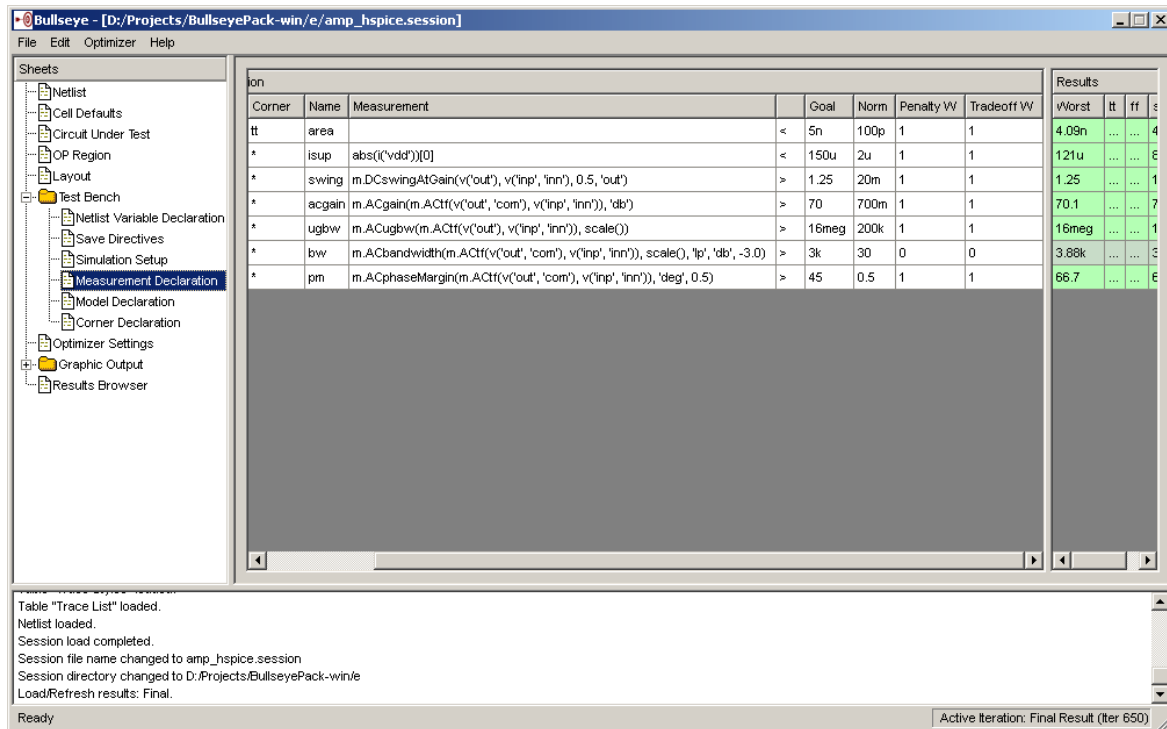


Figure 13: Setting the goals, norms, penalties, and tradeoffs.

For every measurement one of the following goal types can be set in the unnamed column between Measurement and Goal:

- < smaller than goal
- > greater than goal
- <> within tolerance of goal

The goal value is specified in the Goal column. It is a single value for goal types “smaller than” and “greater than”, and a space separated pair specifying the goal and the tolerance in case of the “within tolerance” goal type. The latter results in optimizer forcing the measurement to be between goal-tol and goal+tol.

The norm can be specified for every measurement in the Norm column. Of norm is not specified it is equal to the goal value or 1 if the goal is 0. The penalty and the tradeoff weight are specified in the Penalty W and Tradeoff W columns. The tradeoff weight can be omitted which results in its default value (0).

The values of measurements across all corners along with the worst measurement value for the active iteration selected in the Results Browser can be viewed in the right sub pane of the Measurement Declaration pane. Green fields denote measurements that satisfy the goal while red fields mean that a measurement fails to satisfy the goal. Measurements with both penalty and tradeoff weight set to 0 do not affect the cost function. The coloring of such measurements is a grey shade of green and red.

BULLSEYE Manual

Front-end Design Management

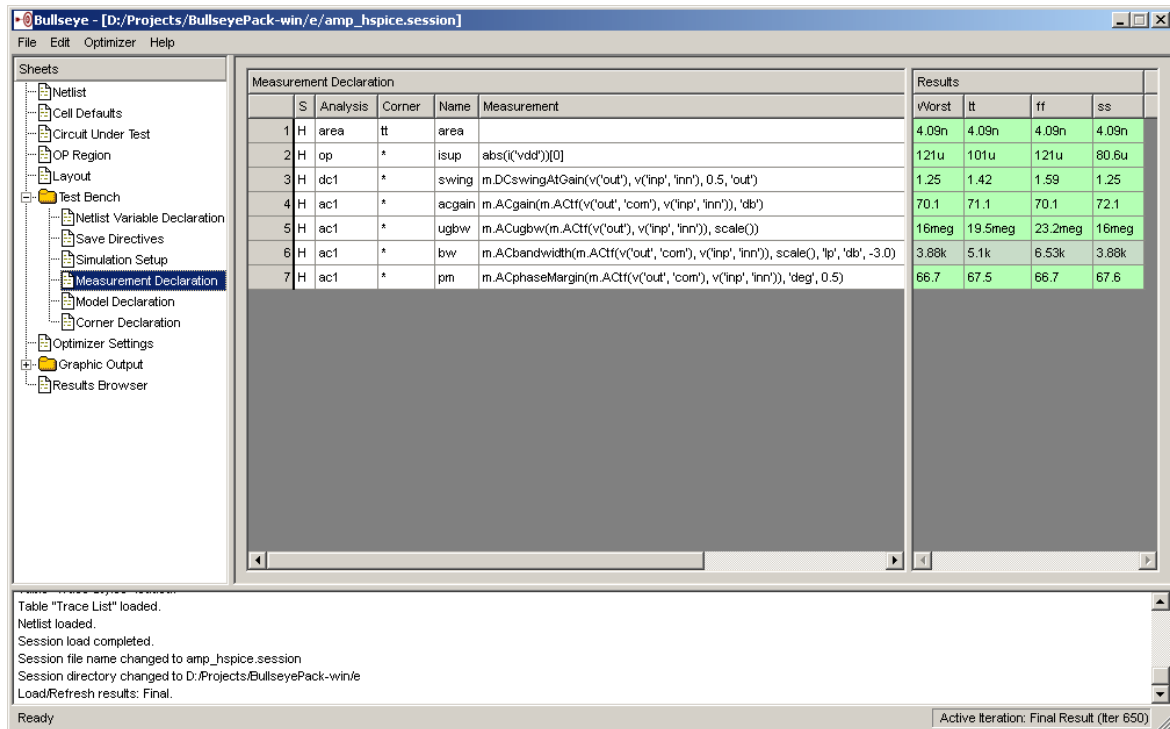


Figure 14: Viewing the measurement results across corners.

BULLSEYE Manual

Front-end Design Management

5. Model declaration

The Model Declaration settings group enables you to declare pairs of the form (file, section) and name them for the purpose of using them as corner models. Every row describes one such pair. The name by which the pair is referred to in the Corner Declaration table is entered in the Name column.

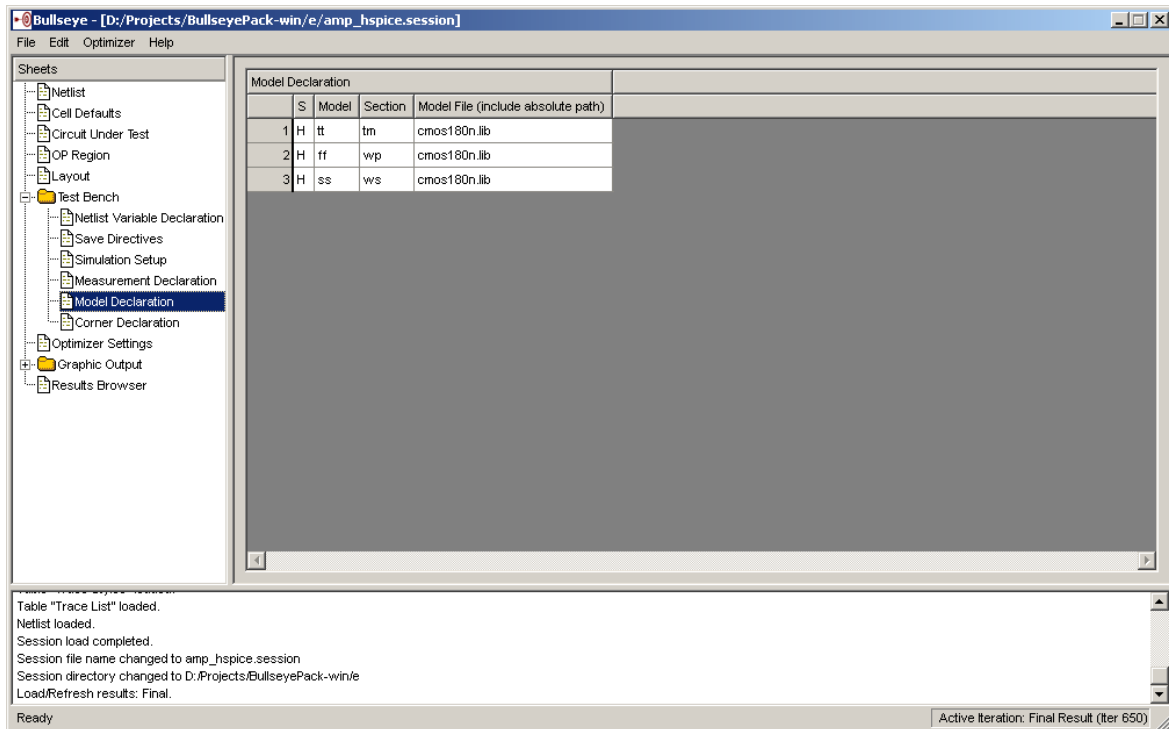


Figure 15: Declaring models.

BULLSEYE Manual

Front-end Design Management

6. Corner declaration

Corners are declared in the Corner Declaration settings group. Every row declares one corner. The name of the corner is entered in the Name column. Model column specifies the name of the (file, section) pair that is used as the model for that corner. The Temp column specifies the temperature for the corner. All variables marked as corner dependent in Variable Declaration settings group appear in the right sub pane of the settings pane. Their value must be defined for every corner.

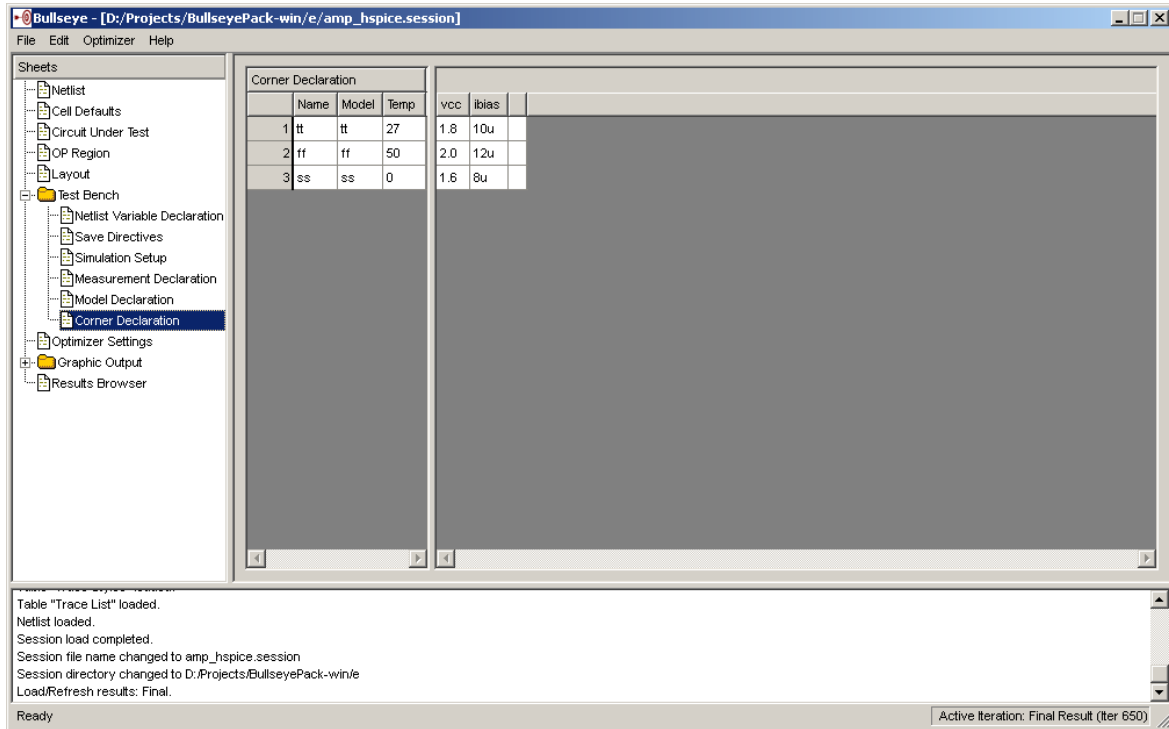


Figure 16: Declaring corners.

IX. Optimizer settings

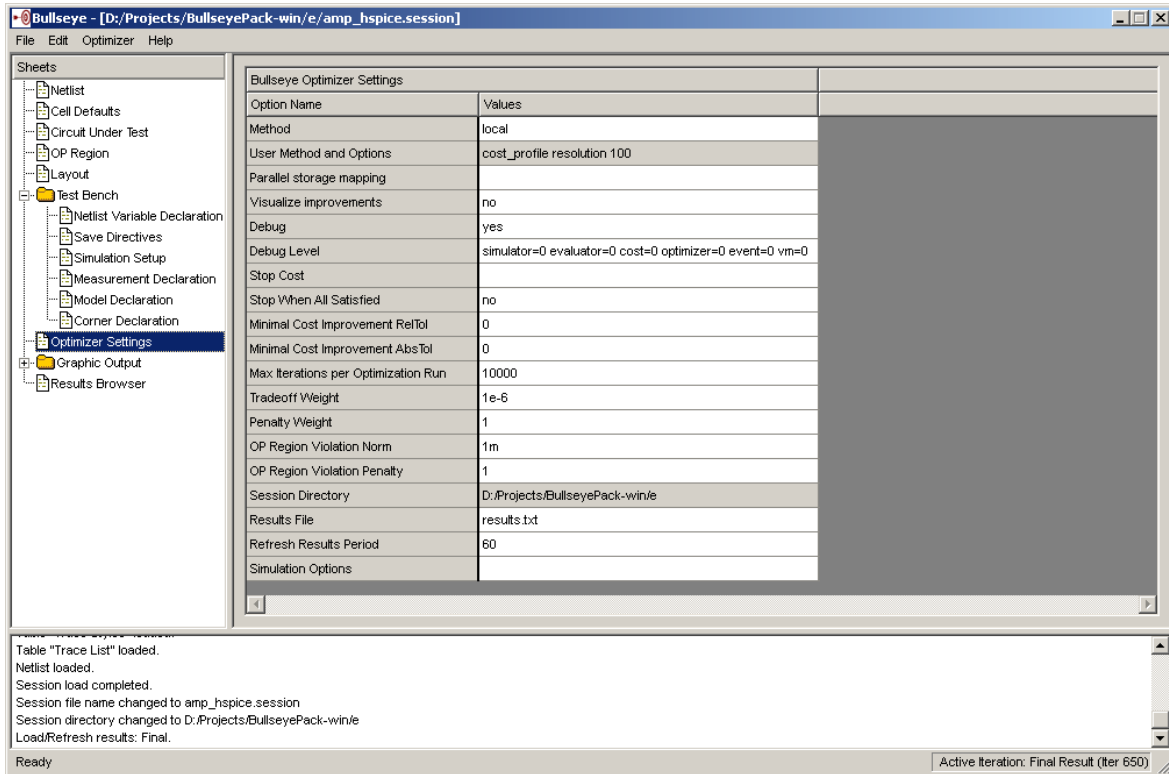


Figure 17: Optimizer settings.

The following optimizer settings are available:

- **Method**

Specifies the optimization method. Available values are

- local
Local method, works with one CPU.
- heuristic
A mixture of local search and random initialization, works with one CPU.
- global
Global method based on simulated annealing and differential evolution.
Can run in parallel on multiple CPUs.
- user defined
Reserved for future use.

- **User Method and Options**

Reserved for future use. Active only when 'user defined' method is selected.

- **Parallel Storage Mapping**

Specifies the mapping of local files and directories to remote files and directories in parallel optimization runs. A mapping is of the form local_name=remote_name. Multiple mappings must be space-separated.

BULLSEYE Manual
Front-end Design Management

- **Visualize Improvements**

If enabled the properties of every circuit that improves the cost function will be plotted according to settings in the Graphic Output settings group.

- **Debug**

Enables debug mode. Debug information goes to the file named log.txt in the session directory.

- **Debug Level**

Debug levels for various modules in the optimizer. A debug level is given as module=value. A higher value means more verbose output. Debugging of a module is turned off if the value is set to 0. If no debug level is specified for a module the debugging is also turned off. Debug levels for multiple modules can be set by separating them with space. The following modules are available:

- simulator - controls the simulator (generates simulator input files, runs analyses and collects results),
- evaluator – evaluates measurements from results collected by the simulator module,
- cost – evaluates the cost function from measurement values produced by the evaluator,
- optimizer – generates optimization parameter values depending on the results obtained from cost function evaluations,
- event – handles event dispatching in parallel optimization algorithms,
- vm – interface to the library taking care of the communication between individual processes in a parallel optimization run (currently only PVM is supported).

- **Stop Cost**

The optimization is stopped if the cost function value becomes smaller than this setting. If no value is specified the optimization runs until some other condition stops it.

- **Stop When All Satisfied**

Stops the optimizer when all measurements fulfill their respective goals. Should not be enabled you want to optimize beyond goals (when tradeoff weights are not all equal to 0).

- **Minimal Cost Improvement RelTol**

Relative tolerance of minimal cost function improvement that triggers the output of full details for an iteration of the optimization run.

- **Minimal Cost Improvement AbsTol**

Absolute tolerance of minimal cost function improvement that triggers the output of full details of an iteration of the optimization run into the results file.

Storing the details in the results file also triggers performance visualization if the Visualize Improvements option is enabled.

If both AbsTol and RelTol are set to 0 all improvements to the best-yet cost function value are stored in the results file.

- **Max Iterations per Optimization Run**

Maximal number of cost function evaluations. After this number is exceeded the optimizer stops. Must be given for all optimization algorithms.

- **Tradeoff Weight**

Global multiplier for tradeoff weights of measurements. A reasonable default is 1e-6 (i.e. 1u).

- **Penalty Weight**

Global multiplier for penalty weights of measurements. A reasonable default is 1. Should be much greater than the value specified as the Tradeoff Weight.

BULLSEYE Manual
Front-end Design Management

- **OP Region Violation Norm**
Norm for operating region constraints. $1e-3$ (i.e. 1m) is a reasonable default.
- **OP Region Violation Penalty**
Penalty weight for operating region constraint violations. A reasonable default is 1.
- **Session Directory**
This cell cannot be written. Instead of that it displays the directory of the current session. This is also the directory where the results file and the log.txt file are written.
- **Results File**
Name of the file summarizing the results of the optimization run. Usually results.txt.
- **Refresh Results Period**
Period in seconds that is used for Results Browser refreshing. A manual refresh can be achieved by selecting Optimizer/Refresh Results in the main menu or by pressing CTRL+R.
- **Simulation Options**
A space separated list of simulator options (.option statement) specified in name=value form.

X. Graphic output setup

1. Graph list

Every row in the Graph List table represents one graph window. The name of the window is specified by the Name column. Width and Height specify the size of the window in pixels. The grid is specified by the Grid column (linear, xlog, ylog, loglog, or polar). If specified, X Limit and Y Limit disable the automatic axis scaling. A fixed range specified in the corresponding cell is used. The range is specified as a set of two space-separated values. Every graph window can have a title, x-axis label and y-axis label (Title, X Label, and Y Label columns).

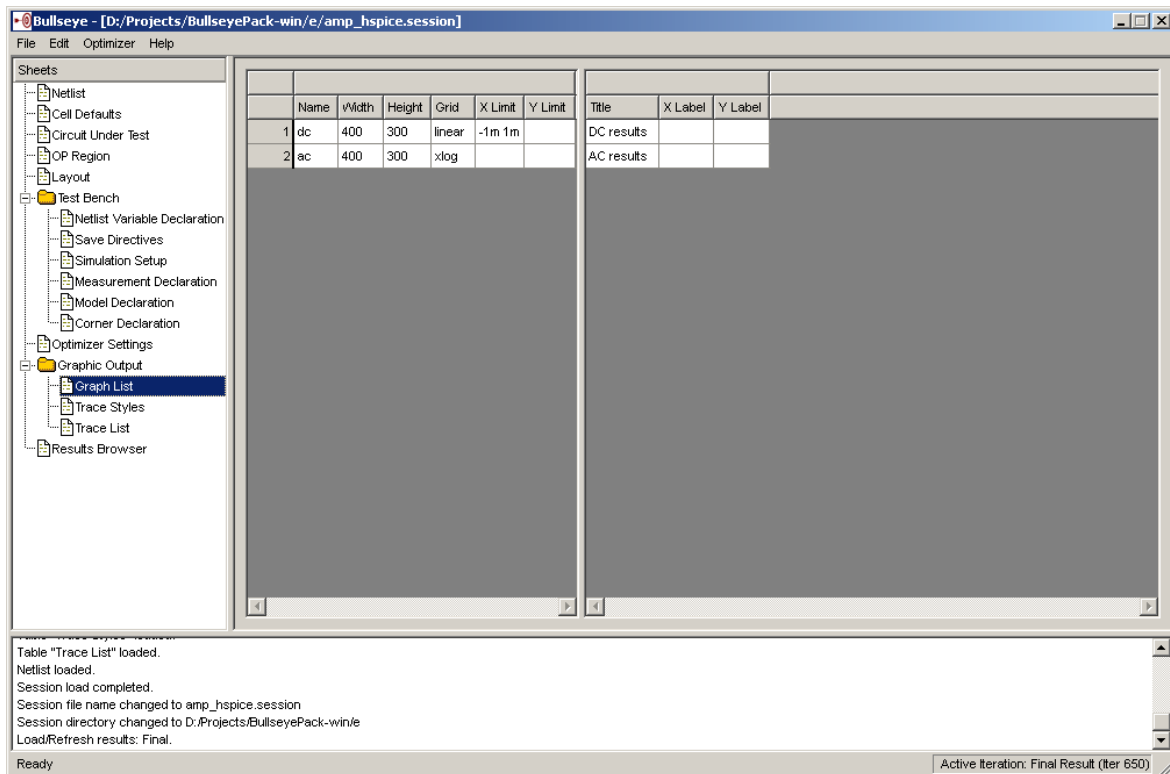


Figure 18: Defining graphic windows.

2. Trace styles

Every row in the Trace Styles table specifies one trace style. Trace styles are tested against trace and corner names. If a match is found the style is applied to the matched trace. Matching is performed against the value of the Name and Corner columns. Either of them can be set to * which matches any name. Trace styles are tried in the same order as they appear in the table (from top to bottom). If multiple styles match a particular trace-corner combination, the last one defines the style used for the trace.

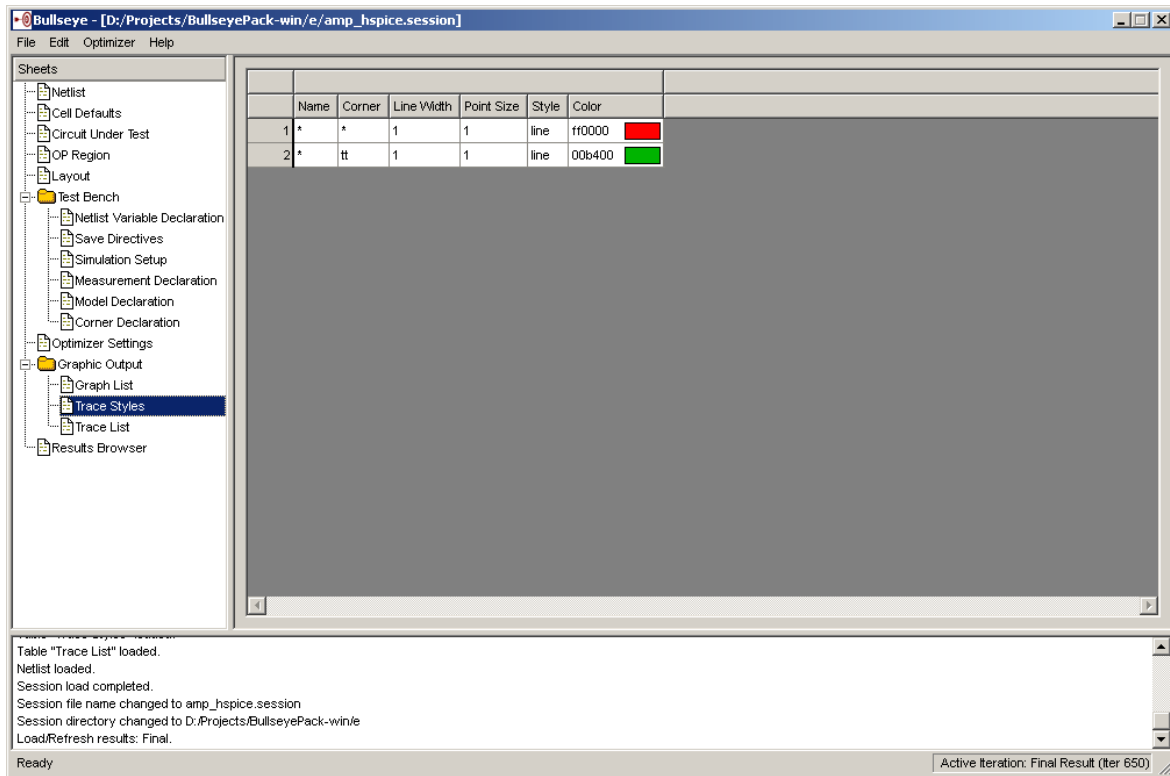


Figure 19: Defining trace styles.

Every trace style specifies the line width, the point size, the plotting style (o, x, +, d, line, or comb), and the color of a trace. Plotting styles o, x, +, and d result in point plots with circular, crosshair, plus, and diamond points, respectively. The colors are specified as 6-digit hex codes where pairs of digits specify the red, green, and the blue component of a color. By double-clicking on a color cell a color picker is opened where you can pick or mix a color with the mouse.

BULLSEYE Manual

Front-end Design Management

3. Trace list

In the Trace List table every row corresponds to one trace (a pair of two expressions specifying the values for the x-axis and the y-axis. The expressions for both axes are specified in the Expression and Scale columns. Every trace has a name specified by the Name column. The Graph column specifies the graph window in which the trace will appear. Analysis column specifies the analysis that supplies the results for the x-axis and the y-axis expressions. Finally Corners specifies a space-separated list of corners for which the trace will be plotted. An asterisk (*) matches all defined corners.

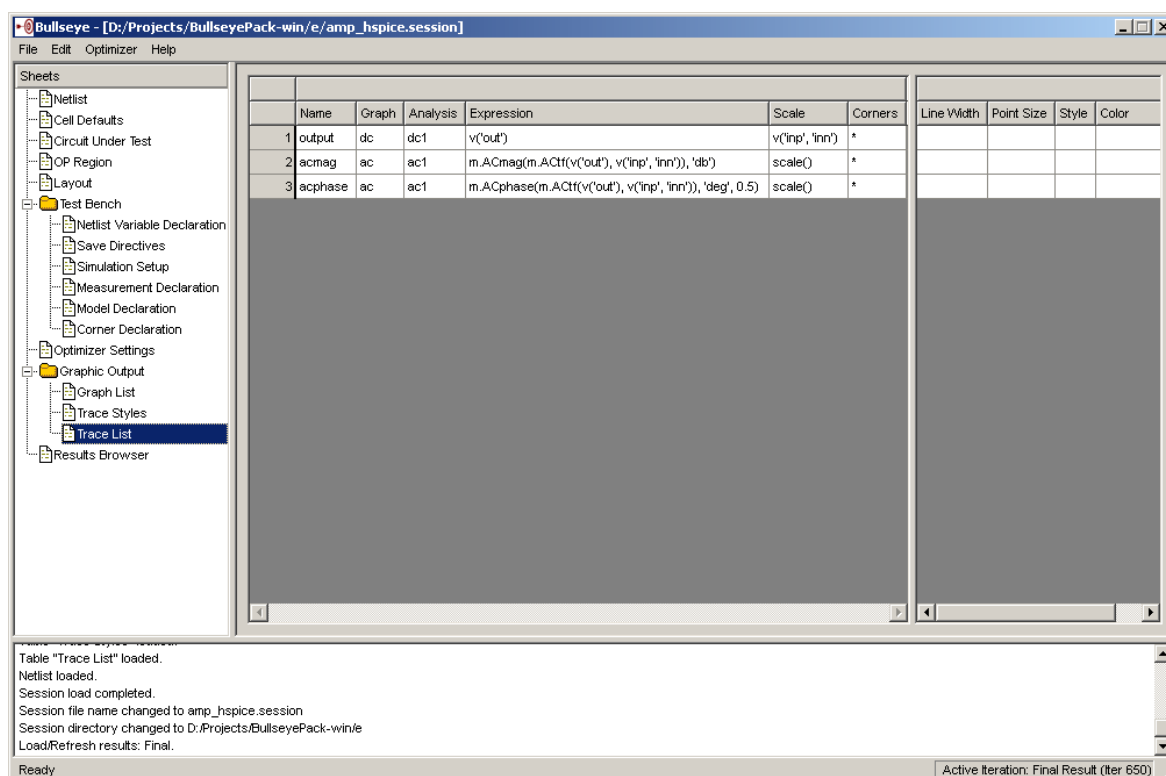


Figure 20: Specifying traces for plotting.

In the right sub pane the trace style specified by the patterns in the Trace Styles table can be overridden (line width, point size, plotting style, and color).

BULLSEYE Manual

Front-end Design Management

XI. Starting a run

The session is exported by selecting Optimizer/Export in the main menu. Errors that occur during the export are listed in the message pane. A run is started by selecting Optimizer/Export And Run in the main menu. This option first exports the session and if the export is successful starts the optimization. An optimization run in progress can be recognized from the console window that pops up. Closing the window stops the optimization.

The results are written into the results file (usually results.txt). The log of the optimizer’s actions and the debug messages are written to the log.txt file. Only one optimization run may be started in a session directory. If you start multiple runs the results are unpredictable.

XII. Inspecting the results (results browser)

As the optimization progresses the results start pouring in. They can be obtained from the results file. A convenient representation of the results can be viewed in the Results Browser. Results get refreshed automatically from time to time (see the Refresh Results Period optimizer setting). Usually you want to refresh results manually by selecting Optimizer/Refresh Results from the main menu or by pressing CTRL+R.

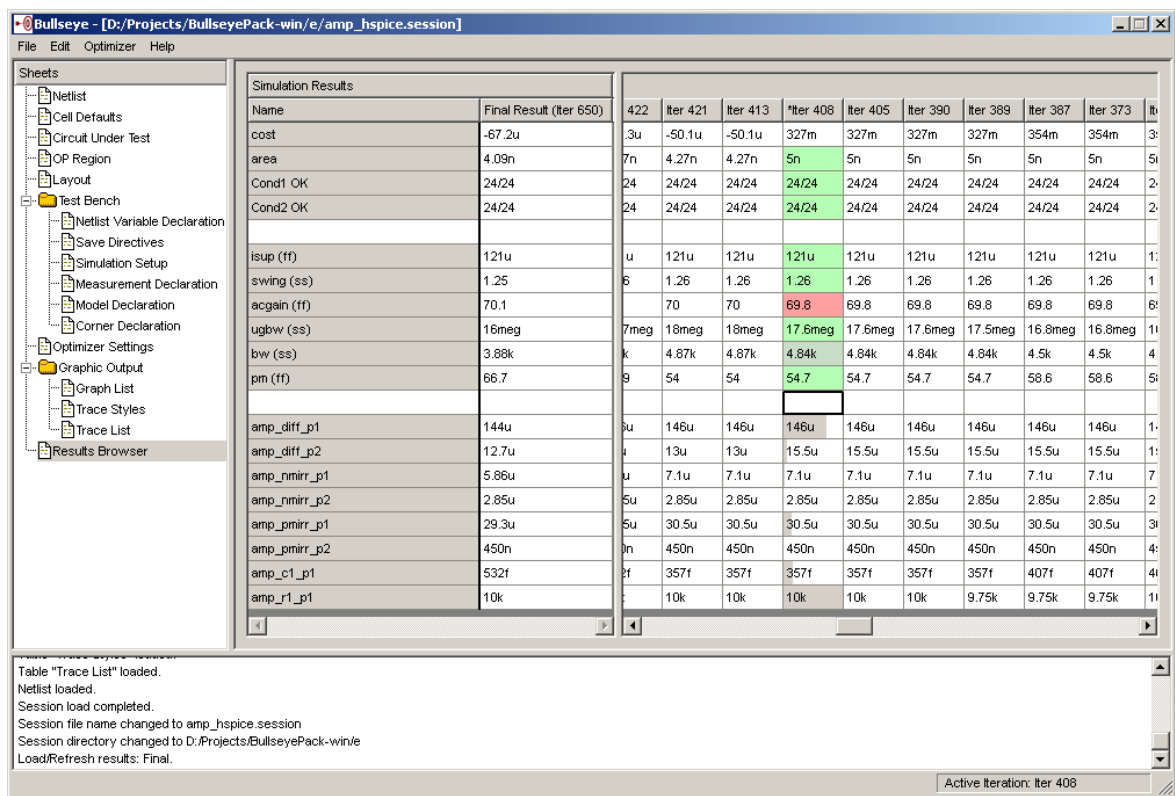


Figure 21: Results browser with the last iteration not satisfying all goals marked active.

The right sub pane of the results browser displays the results of one iteration per column. The left sub pane displays the results of the best iteration. Iteration can be made active by clicking on its column. The

BULLSEYE Manual

Front-end Design Management

active iteration is displayed in the bottom right corner of the Bullseye window. Details of the active iteration are available in the Circuit Under Test, OP Region, and Measurement Declaration tables.

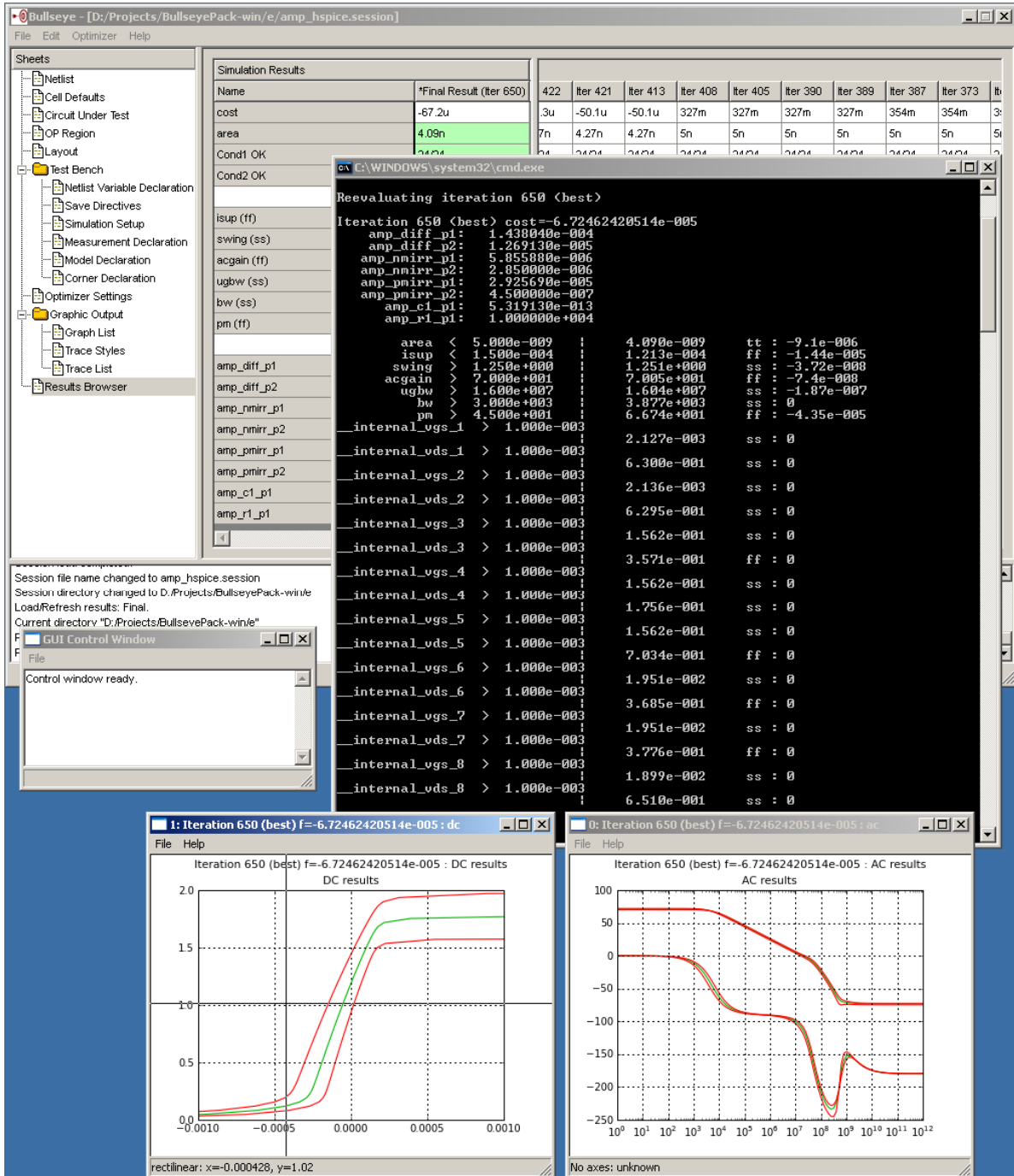


Figure 22: Displaying the properties of an iteration graphically.

For every iteration the corresponding values of optimization parameters are displayed along with bars indicating the relative position of parameter values with respect to the minimal and maximal allowed value on the bottom of every column. The middle part of every column displays the worst values of

BULLSEYE Manual

Front-end Design Management

measurements. In the Name column the corner where the worst value occurs is displayed along with the measurement name for the active iteration.

At the top the iteration number, the corresponding cost function value, and the area of the circuit corresponding to the iteration are displayed. Cond1 OK and Cond2 OK display the number of satisfied operating region requirements against the total number of operating region requirements.

Color of a cell indicates if the worst measurement value satisfies the goal. Red means that the measurement fails to satisfy the goal while green means that the measurement satisfies the goal. If the both tradeoff and penalty weight of a measurement are set to zero the measurement does not affect the cost function value. Such measurements are color coded using a gray shade of red and green.

By right-clicking a cell the context menu of a column (iteration) is opened. Selecting the Set as Seed option copies the values of the optimization parameters corresponding to the clicked iteration into the Seed column of the Circuit Under Test table.

By selecting the Plot Iteration option from the context menu the iteration is reevaluated in the simulator and the corresponding graph windows (the ones defined in the Graphic Output settings group) for that iteration are displayed. The progress of evaluation and its results are displayed in a console window.

Plotting of iteration may be invoked while the optimization is running albeit it may appear slow due to the workload the optimization imposes on the CPU.

Plot windows can be closed by closing the console window that displays the simulation results of the selected iteration. Do not close the console window that popped up when the optimization was started because closing it will stop the optimization.