Pentode
Power Tree Designer

A tool to simulate,
draw and plot
electrical power trees

By Simone Pernice

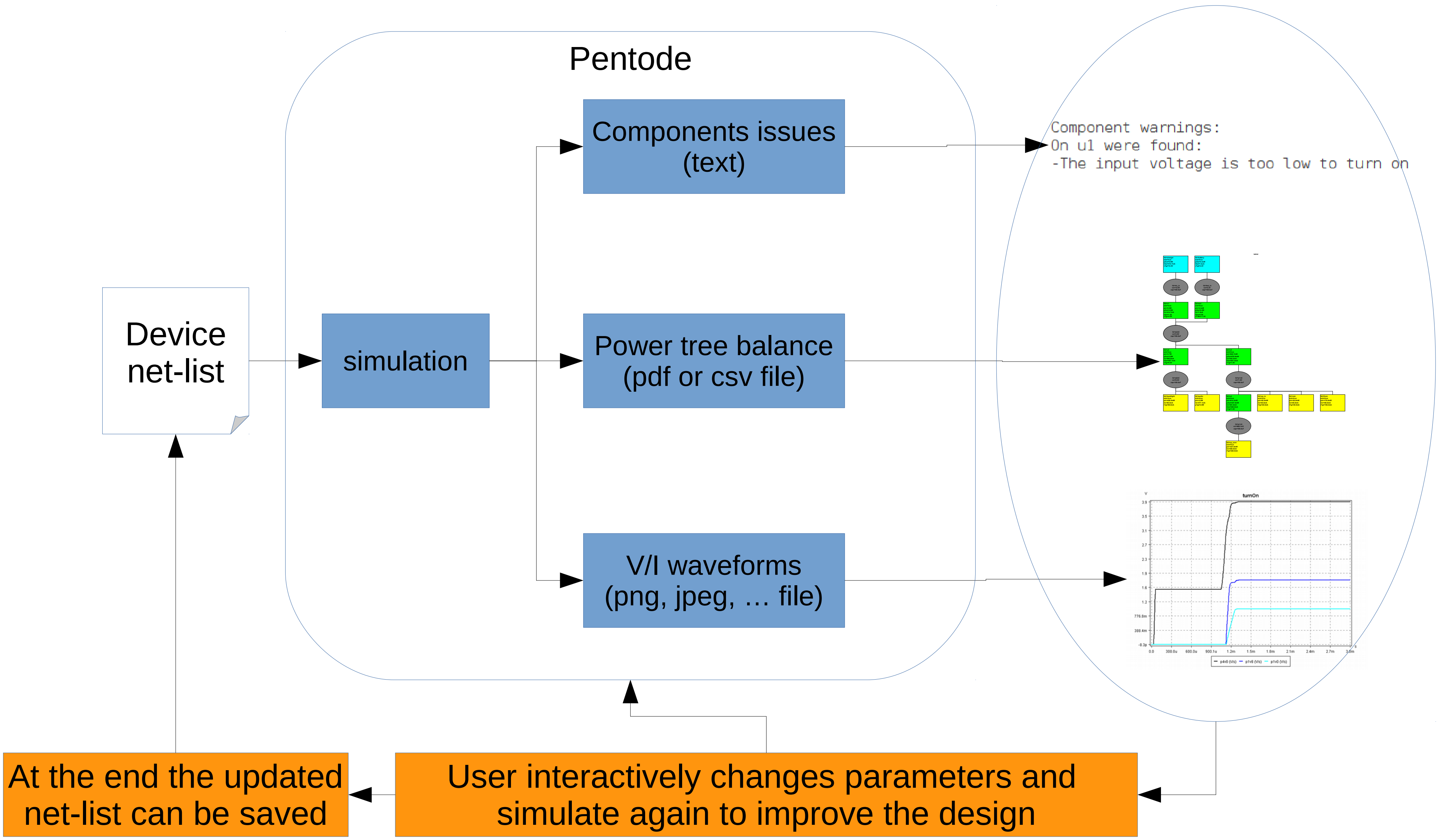
Power tree design tools

- A lot of tools are available to simulate analog circuits, power converters, and digital circuits
- Very few tools (if any) are available to design and simulate the whole high level power section of a device
- Pentode, given a simple high level net list of the device, is able to:
  - Simulate the voltage and current up to the steady state
  - Show components working out of specification
  - Draw a nice power tree diagram showing the currents/powers balance
  - Plot node transient voltage and gate current waveforms
  - Change component parameters interactively to improve the design

Power tree design using Pentode

- Pentode takes as input a net list containing:
  - A description of the high level power architecture. Usually one line of text is enough to describe one component of the device
  - Commands to execute on the power architecture
- Pentode executes commands to:
  - Simulate the device reporting any component working out of specification (over voltage at regulator output, diode reverse biased, etc.)
    - Simulate stepping parameters looking for the best performance
  - Draw a (nice to see) power tree diagram on pdf/csv file
  - Plot voltage/current versus time (or stepped parameter) waves on png/jpeg file
  - Execute loops, iterate though lists, check conditionals and perform basic calculus to automate repetitive task
- In the net-list commands or interactively, it is possible to:
  - Change the components parameters (enable, quiescent current, etc.)
  - Run new simulations to verify the performance of the system for all the use cases
- It is possible to change the net-list to describe a different power tree
  - It takes few minutes to repeat the tests on all the use cases

Pentode work flow


Pentode
Device net-list
simulation
Components issues (text)
Power tree balance (pdf or csv file)
V/I waveforms (png, jpeg, … file)
turnOn
At the end the updated net-list can be saved
User interactively changes parameters and simulate again to improve the design

```
Component warnings:
On u1 were found:
-The input voltage is too low to turn on
```

Power tree requirement

The main concerns designing the power tree of a new device are:

- Efficiency: it is good to minimize the input power especially on portable devices to provide long battery life. High efficiency can be achieved with:
  - Good power tree structure and converters selection
  - Scalability: the capacity to turn off the not used regulators in order to reduce the energy waste
  - Standby current: is the special case of scalability where almost all components are turned off and the device has to preserve the battery draining a negligible current drain
- Cost: the device has to be the cheapest possible but that goes against efficiency because usually more efficient converters are more expensive
- Ratings: every source, converter and drain has to work within its specification
- Correct suppliers turn on and off sequence: that is very important when the components linked together use several rails

Traditional power tree design

Usually the power tree designer requires to:

- Define all the user cases:
  - Standby, WiFi, Video + WiFi, Video + Ethernet, ...
- Define the most suitable power trees
  - That is a trial and error procedure which requires to draw on paper the most promising architectures
  - Compute the cost of every architecture
- For each couple: use case – architecture the designer has to evaluate the efficiency computing the power balance
- Choose the best trade off between cost and efficiency

Pentode can automate the procedure above

There are few tools available to automate that procedure

  - For instance it can be done on a standard spreadsheet. However that requires to rewrite the power balance equations for each architecture evaluated. Moreover it is difficult to use complex mathematical relationship to describe second order effects like the switching converter efficiency which decrease at very low output current or the leakage current on a LDO

Pentode components

- Pentode simulates a net of linked electric components
- Pentode component can be:
  - Node, defined by:
    - voltage and capacitance
  - Black box, defined by:
    - gates which are linked to the previous nodes
    - the current flowing on those gates are defined by not-linear time-variant behavioral equations depending on the voltage of the nodes at which they are linked to
    - the user set the behavior of those equation defining the component parameters like the output voltage of a regulator

Black boxes

- The black boxes can be:
  - Sources:
    - Have just the output gate
    - Provide power to the device like a DC adapter or a battery
    - The behavioral model takes into account of parameters like:
      - Internal resistance, maximum current, voltage waveform (for time variant source), etc.
  - Converters:
    - Have input and output gates and may have also a control gate
    - Transfer the power from input to output gate like diodes, switches, LDOs, buck or boost converters
    - The behavioral model takes into account of parameters like:
      - Load and line regulation, maximum current, minimum input voltage, quiescent and disable current; for switching converters: switching and resistive current loss, working mode (PWM or PFM), etc.
  - Drains:
    - Have just the input gate
    - Use the power provided by sources and converters like LED or resistor
    - The behavioral model takes into account of parameters like:
      - Minimum operating voltage, equivalent resistance, current or power, etc.

Component definition

- The Pentode components are defined in the net-list as follows:

```
Name Label Node1 NodeN Property1 = value1 PropertyN = valueN
```

- Name is the type of component like for instance:
  - **srcVoltage** for a voltage source
  - **cnvLDO** for a LDO converter
  - **drnPower** for a constant power drain
- Label is the unique name given to this device
- Node1 to NodeN are the nodes a which is linked the device
- Propery1 is a property of the device set to value1.
  - Devices have a lot of property with default value.
  - The not-set properties are kept at default value.
  - The user can just set the property to be modified respect to default

Embed branded converters

- In order get an accurate simulation the user is required to set the converter parameters. To find those parameters the user has to:
  - Look at the data sheet for the basic converter figures
  - Simulate the converter on Pentode and tuning its parameters in order to match the data sheet graphs if available or an evaluation board behavior if the data is not reported
- That procedure is long and tedious
- However it would be possible to store one brand of converters on Pentode
  - The user would simply play with them them ignoring most of their behavioral parameters
  - That will help the user to design quickly its device based on that brand of components

Commands

- The Pentode commands works on the current net-list. Their syntax recall the component definition:

```
Name Label Property1 = value1 PropertyN = valueN
```

- Name is the command name like:
  - **simulateSteady** to simulate the current device up to the steady state
  - **draw** to draw a power tree
  - **plot** to plot a voltage or current
  - **help** to print a command function, properties and syntax
- Label it is used in some commands, for instance on **plot** and draw is the output file name
- Propery1 is a property of the command, for instance **tEnd** is the end time of a simulation

Net list example

- The following is a simple Pentode net list describing:
  - A voltage source labeled charger linked at dc_in node providing 12V
  - A buck converter labeled u1 linked from dc_in to p4v0 to provide 4V output and able to sustain 15V reverse
  - A voltage source labeled battery linked at bat_in node providing 4.2V
  - A boost converter labeled u2 linked from p4v0 to p8v0 to supply back light and audio
  - A switch labeled sw1 linked from bat_in to p4v0 active when the node dc_in is between -1 and 4V
  - A constant current drain labeled backlight draining 80mA for back light
  - The following lines are self explaining...

```
srcVoltage      charger         dc_in                       vLow =  12V
cnvBuck         u1              dc_in       p4v0            vTgt = 4V    vRev = 15V

srcVoltage      battery         bat_in                      vLow = 4.2V
cnvCntSwitch    sw1             bat_in      p4v0    dc_in   vTrMn = -1V vTrMx=4V

cnvBoost        u2              p4v0        p8v0            vTgt = 8V
drnCurrent      backlight       p8v0                        iTgt = 80mA
drnPower        audio           p8v0                        pTgt = 3W

cnvBuck         u3              p4v0        p1v8            vTgt = 1.8V  vMin = 3V

drnCurrent      up_io           p1v8                        iTgt = 30mA
drnCurrent      ram             p1v8                        iTgt = 20mA
drnCurrent      from            p1v8                        iTgt = 100mA

cnvLDO          u4              p1v8        p1v0            vTgt = 1.0V
drnCurrent      up_core         p1v0                        iTgt = 350mA    vMin = 0.5V
```

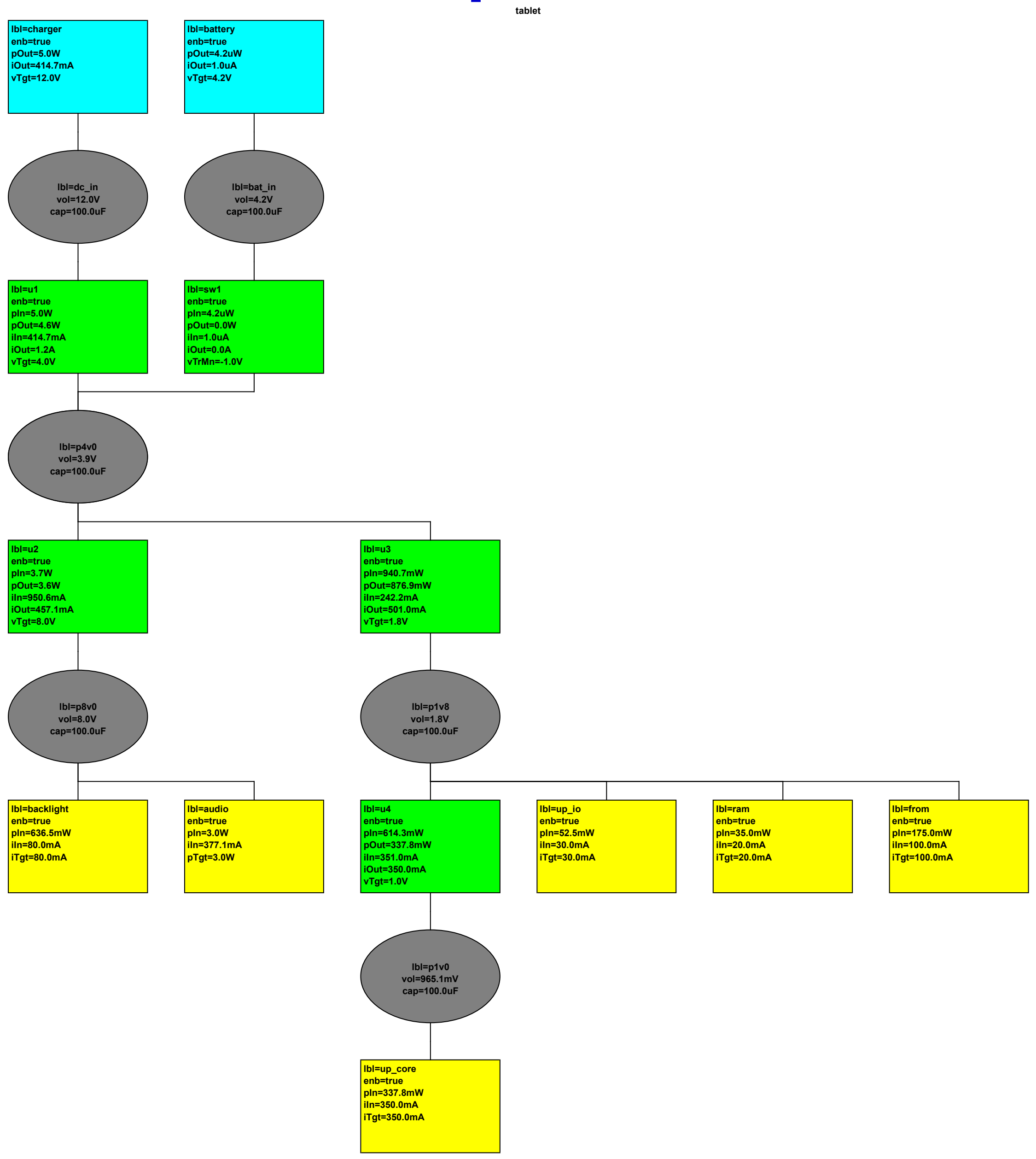
Net list draw example

- The previous net-list is simulated during transient up to 3ms with following command:

```
simulateTransient tEnd = 3ms
```

- The power tree is then drawn and saved on tablet.svg file with following command:

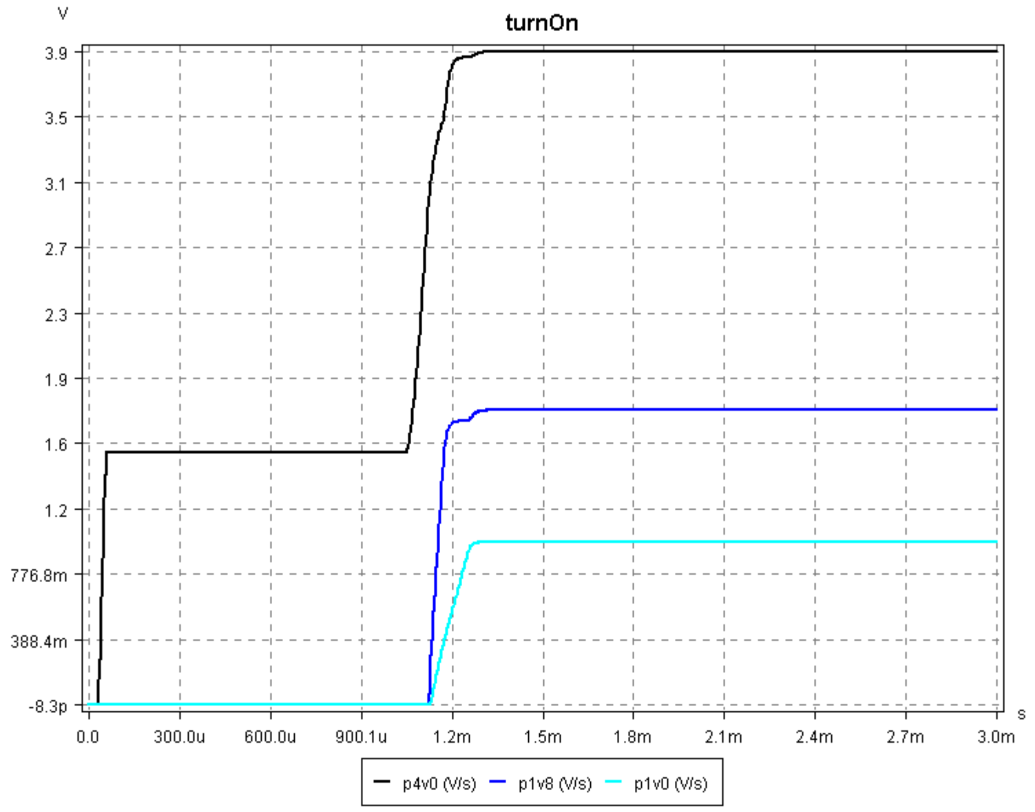
```
draw tablet outputFormat = svg
```


tablet
lbl=charger enb=true pOut=5.0W iOut=414.7mA vTgt=12.0V
lbl=battery enb=true pOut=4.2uW iOut=1.0uA vTgt=4.2V
lbl=dc_in vol=12.0V cap=100.0uF
lbl=bat_in vol=4.2V cap=100.0uF
lbl=u1 enb=true pIn=5.0W pOut=4.6W iIn=414.7mA iOut=1.2A vTgt=4.0V
lbl=sw1 enb=true pIn=4.2uW pOut=0.0W iIn=1.0uA iOut=0.0A vTrMn=-1.0V
lbl=p4v0 vol=3.9V cap=100.0uF
lbl=u2 enb=true pIn=3.7W pOut=3.6W iIn=950.6mA iOut=457.1mA vTgt=8.0V
lbl=u3 enb=true pIn=940.7mW pOut=876.9mW iIn=242.2mA iOut=501.0mA vTgt=1.8V
lbl=p8v0 vol=8.0V cap=100.0uF
lbl=p1v8 vol=1.8V cap=100.0uF
lbl=backlight enb=true pIn=636.5mW iIn=80.0mA iTgt=80.0mA
lbl=audio enb=true pIn=3.0W iIn=377.1mA pTgt=3.0W
lbl=u4 enb=true pIn=614.3mW pOut=337.8mW iIn=351.0mA iOut=350.0mA vTgt=1.0V
lbl=up_io enb=true pIn=52.5mW iIn=30.0mA iTgt=30.0mA
lbl=ram enb=true pIn=35.0mW iIn=20.0mA iTgt=20.0mA
lbl=from enb=true pIn=175.0mW iIn=100.0mA iTgt=100.0mA
lbl=p1v0 vol=965.1mV cap=100.0uF
lbl=up_core enb=true pIn=337.8mW iIn=350.0mA iTgt=350.0mA

Net list plot example

The turn on sequence is plot on the file turnOn.png with the following command:

```
plot turnOn TIME, p4v0, p1v8, p1v0
```



P4v0 is flat for about 1.1ms. P1V8 reaches the final voltage only after p1v0 is ready

That power sequence have to be allowed by the involved component

Net list modify example

- It is also possible to interactively modify components and run again the simulation.

- In the next example the charger is turned off

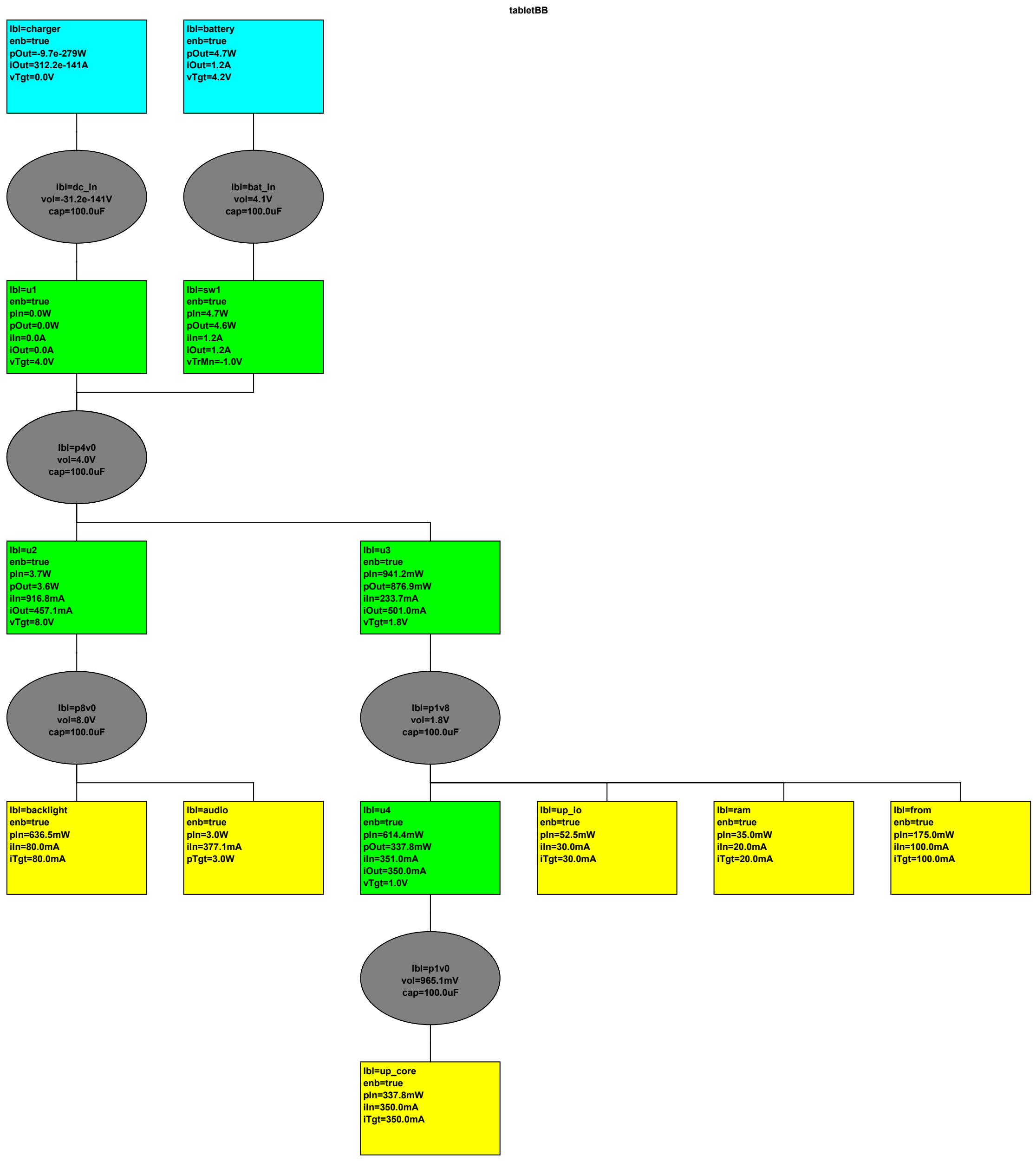
```
modify charger vLow  = 0
```

- Then the simulation is run again:

```
simulateTransient tEnd = 3ms
```

- The new power tree is saved on the tableBB.pf file:

```
draw tabletBB outputFormat = pdf
```


tabletBB
lbl=charger enb=true pOut=-9.7e-279W iOut=312.2e-141A vTgt=0.0V
lbl=battery enb=true pOut=4.7W iOut=1.2A vTgt=4.2V
lbl=dc_in vol=-31.2e-141V cap=100.0uF
lbl=bat_in vol=4.1V cap=100.0uF
lbl=u1 enb=true pIn=0.0W pOut=0.0W iIn=0.0A iOut=0.0A vTgt=4.0V
lbl=sw1 enb=true pIn=4.7W pOut=4.6W iIn=1.2A iOut=1.2A vTrMn=-1.0V
lbl=p4v0 vol=4.0V cap=100.0uF
lbl=u2 enb=true pIn=3.7W pOut=3.6W iIn=916.8mA iOut=457.1mA vTgt=8.0V
lbl=u3 enb=true pIn=941.2mW pOut=876.9mW iIn=233.7mA iOut=501.0mA vTgt=1.8V
lbl=p8v0 vol=8.0V cap=100.0uF
lbl=p1v8 vol=1.8V cap=100.0uF
lbl=backlight enb=true pIn=636.5mW iIn=80.0mA iTgt=80.0mA
lbl=audio enb=true pIn=3.0W iIn=377.1mA pTgt=3.0W
lbl=u4 enb=true pIn=614.4mW pOut=337.8mW iIn=351.0mA iOut=350.0mA vTgt=1.0V
lbl=up_io enb=true pIn=52.5mW iIn=30.0mA iTgt=30.0mA
lbl=ram enb=true pIn=35.0mW iIn=20.0mA iTgt=20.0mA
lbl=from enb=true pIn=175.0mW iIn=100.0mA iTgt=100.0mA
lbl=p1v0 vol=965.1mV cap=100.0uF
lbl=up_core enb=true pIn=337.8mW iIn=350.0mA iTgt=350.0mA

Pentode binary distribution

- Pentode is written in Java
- It is possible run Pentode on a server and return its output files (power trees and waveforms) to the users
- Pentode is based on few GPL libraries to provide output files (pdf, svg, png). Therefore to distribute Pentode binary it is required to provide its source code.
- Those libraries are available also with LGPL license paying their royalties. Therefore it is possible distribute Pentode without its source code if needed.

Library dependences

Pentode uses the following external libraries:

- To draw power tree in pdf file:
  - Orson PDF
  - GPL license can be LGPL paying the royalty
- To draw power tree in svg files:
  - jFreeSVG
  - GPL license can be LGPL paying the royalty
- To plot diagram in png file:
  - simple-java-plot by Yuriy Guskov
  - MIT license
- To print figures in engineer notation:
  - engNotation by Andrew Greensted
  - Free license, just requires to show the library author name