gnucap and related work – development status

Felix Salfelder

FOSDEM 2016

About gnucap – GNU Circuit Analysis Package

- Quick History
  - 1983. First traces (Albert Davis)
  - 1990. *ACS*, Al's Circuit Simulator
  - 1992. GPL
  - 2001. Renamed to *gnucap*, a GNU project
  - 2013. Source repos at `git.savannah.gnu.org`

About gnucap – GNU Circuit Analysis Package

- Quick History
  - 1983. First traces (Albert Davis)
  - 1990. *ACS*, Al's Circuit Simulator
  - 1992. GPL
  - 2001. Renamed to *gnucap*, a GNU project
  - 2013. Source repos at `git.savannah.gnu.org`
- featuring
  - (single engine) mixed signal kernel
  - efficient algorithms (sparse matrix, bypassing, etc.)
  - interactive user interface
  - *modelgen*, a model compiler of the early days
  - a spice wrapper, support for spice-style models (C).
  - lots of semiconductor device models available
  - shared library

gnucap – pluggability

- What is a "Plugin"?
  - *Run time* extension (see `dlopen(3)`)
  - Register to *dispatcher* (dictionary) upon loading

gnucap – pluggability

- What is a "Plugin"?
  - *Run time* extension (see `dlopen(3)`)
  - Register to *dispatcher* (dictionary) upon loading
  - Contributions without forking
  - Avoid combinatorial explosion

gnucap – pluggability

- What is a "Plugin"?
  - *Run time* extension (see `dlopen(3)`)
  - Register to *dispatcher* (dictionary) upon loading
  - Contributions without forking
  - Avoid combinatorial explosion
- Plugin classes
  - Components, models
  - Commands, algorithms
  - Functions
  - Netlist/schematic languages
  - Interactive help

gnucap – pluggability

- What is a "Plugin"?
  - *Run time* extension (see `dlopen(3)`)
  - Register to *dispatcher* (dictionary) upon loading
  - Contributions without forking
  - Avoid combinatorial explosion
- Plugin classes
  - Components, models
  - Commands, algorithms
  - Functions
  - Netlist/schematic languages
  - Interactive help
  - data output (under construction)
  - Nodes (maybe)

gnucap-uf, an experimental fork

- Playground for unstable features/plugins

gnucap-uf, an experimental fork

- Playground for unstable features/plugins
  - More SPICE support and components (`poly(k)`)
  - Some (*IR) filter models
  - Transient noise model

gnucap-uf, an experimental fork

- Playground for unstable features/plugins
  - More SPICE support and components (`poly(k)`)
  - Some (*IR) filter models
  - Transient noise model
  - Sensitivity analysis
  - Wave stash (a dictionary for post processing)
  - Operating point stack
  - Enhanced module loading (compile-on-demand)

gnucap-uf, an experimental fork

- Playground for unstable features/plugins
  - More SPICE support and components (`poly(k)`)
  - Some (*IR) filter models
  - Transient noise model
  - Sensitivity analysis
  - Wave stash (a dictionary for post processing)
  - Operating point stack
  - Enhanced module loading (compile-on-demand)
- Incompatible changes
  - Changes in parameter processing, logic evaluation etc.
  - GNU build system, automated test suite, ...
  - Various fancy ideas, partly half-baked, partly obsolete.
  - API converging back to upstream

gnucap-uf, benefits

- Sometimes

gnucap-uf, benefits

- Sometimes
  - Fixes apply to the upstream project

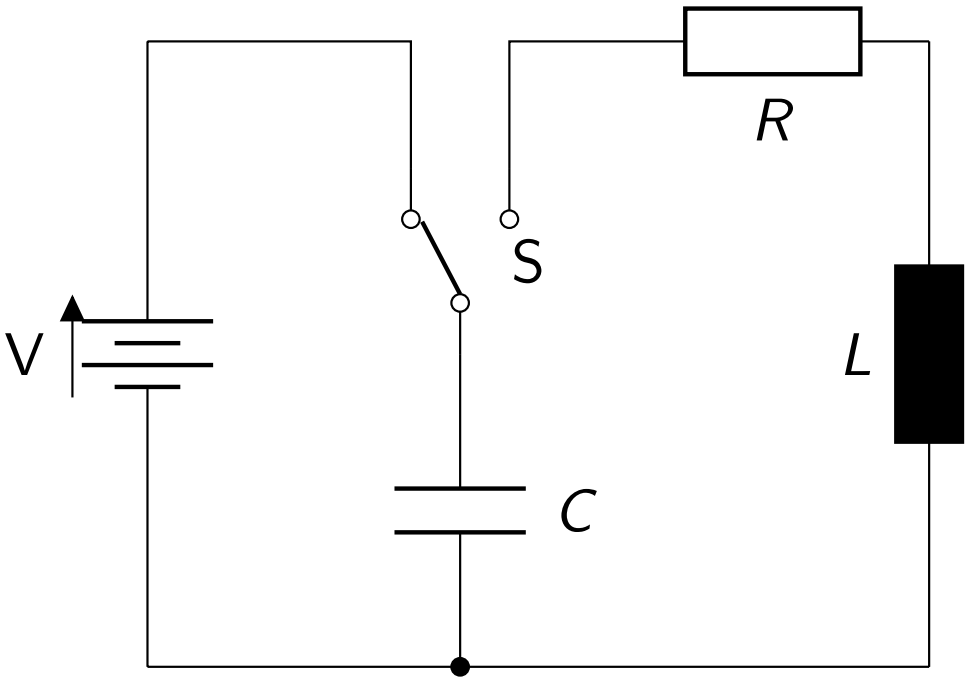
gnucap-uf, benefits

- Sometimes
  - Fixes apply to the upstream project
  - Extensions are portable.

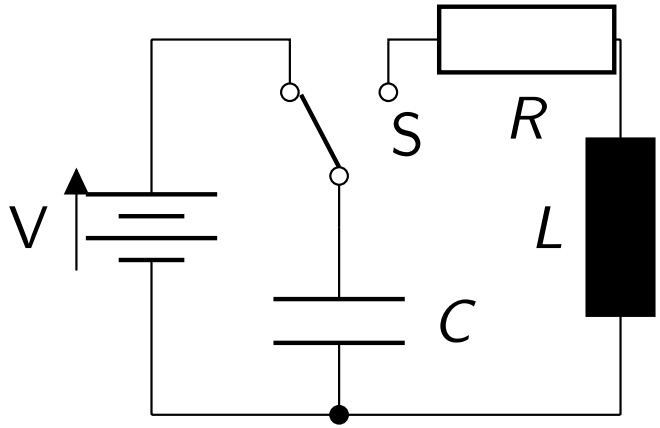
gnucap-uf, benefits

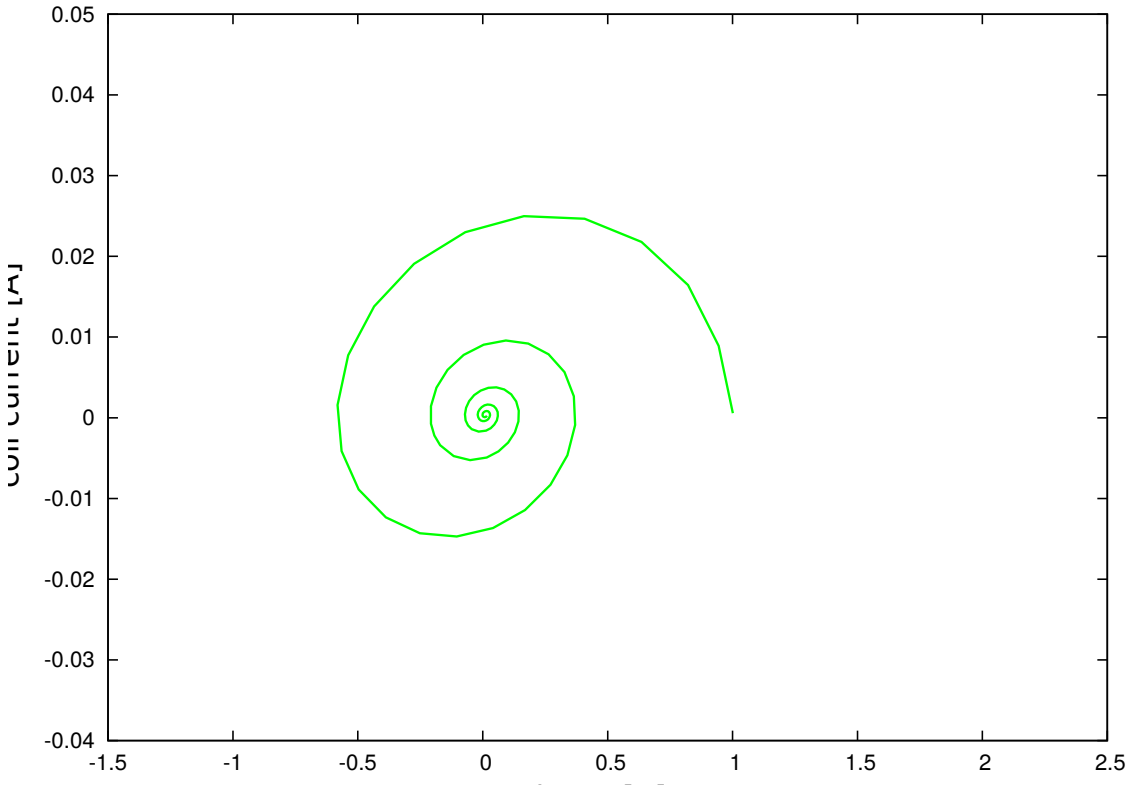
- Sometimes
  - Fixes apply to the upstream project
  - Extensions are portable.
- Platform for some research.
  - State space inspection
  - Ageing effects simulation

gnucap-uf, state space inspection

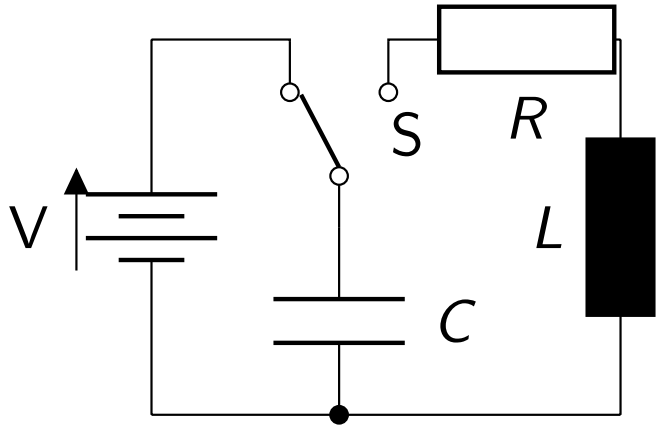


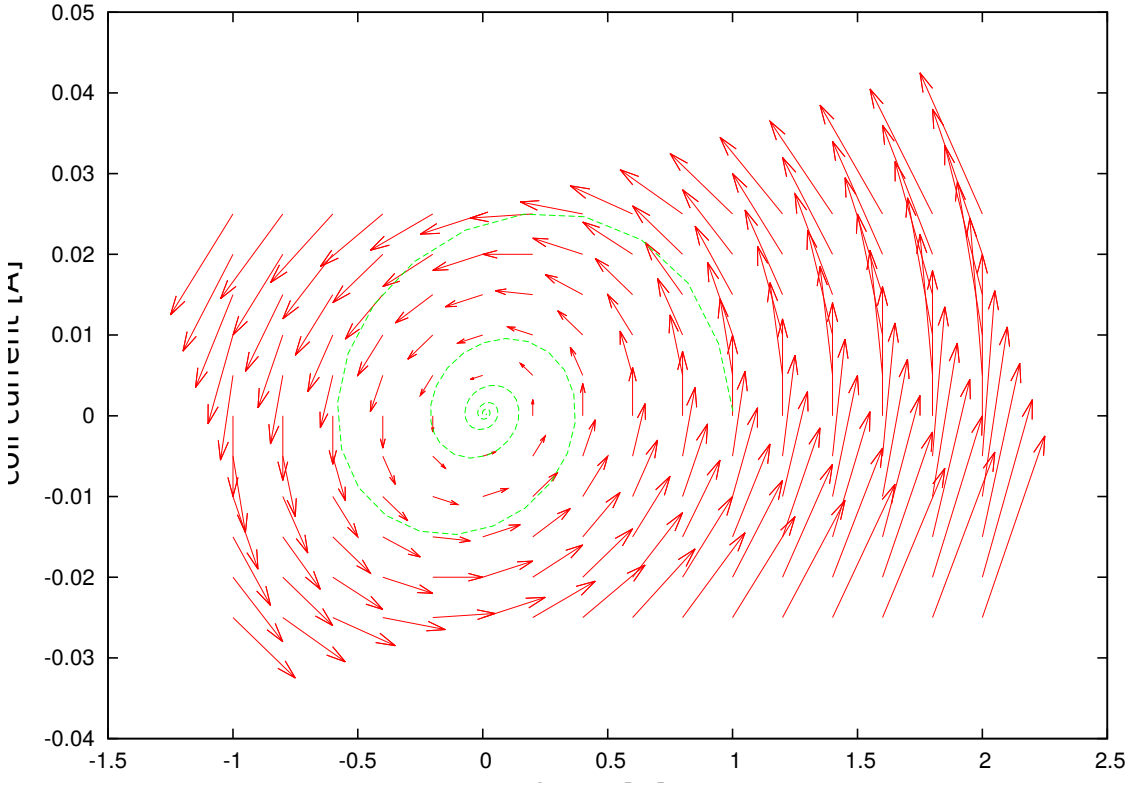
gnucap-uf, state space inspection



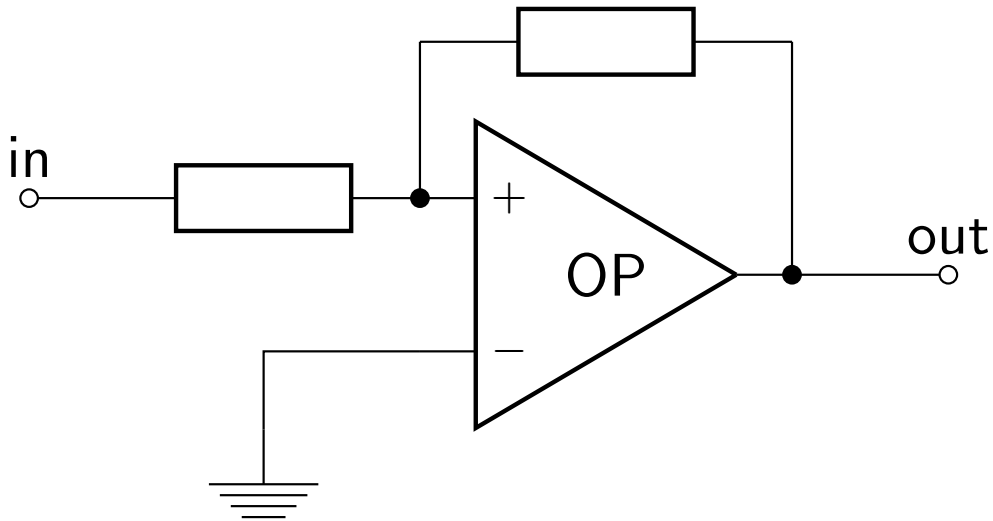


gnucap-uf, state space inspection

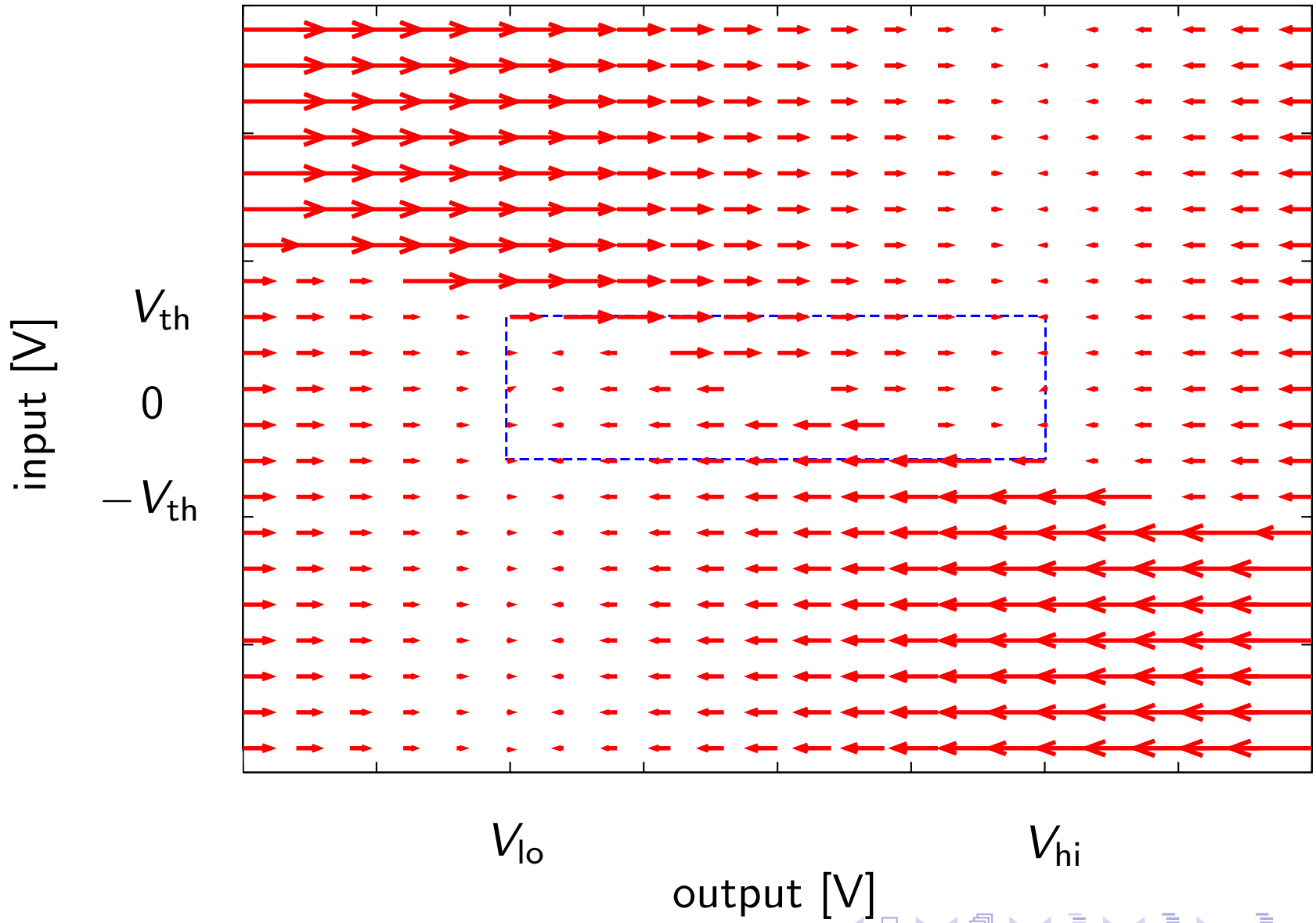




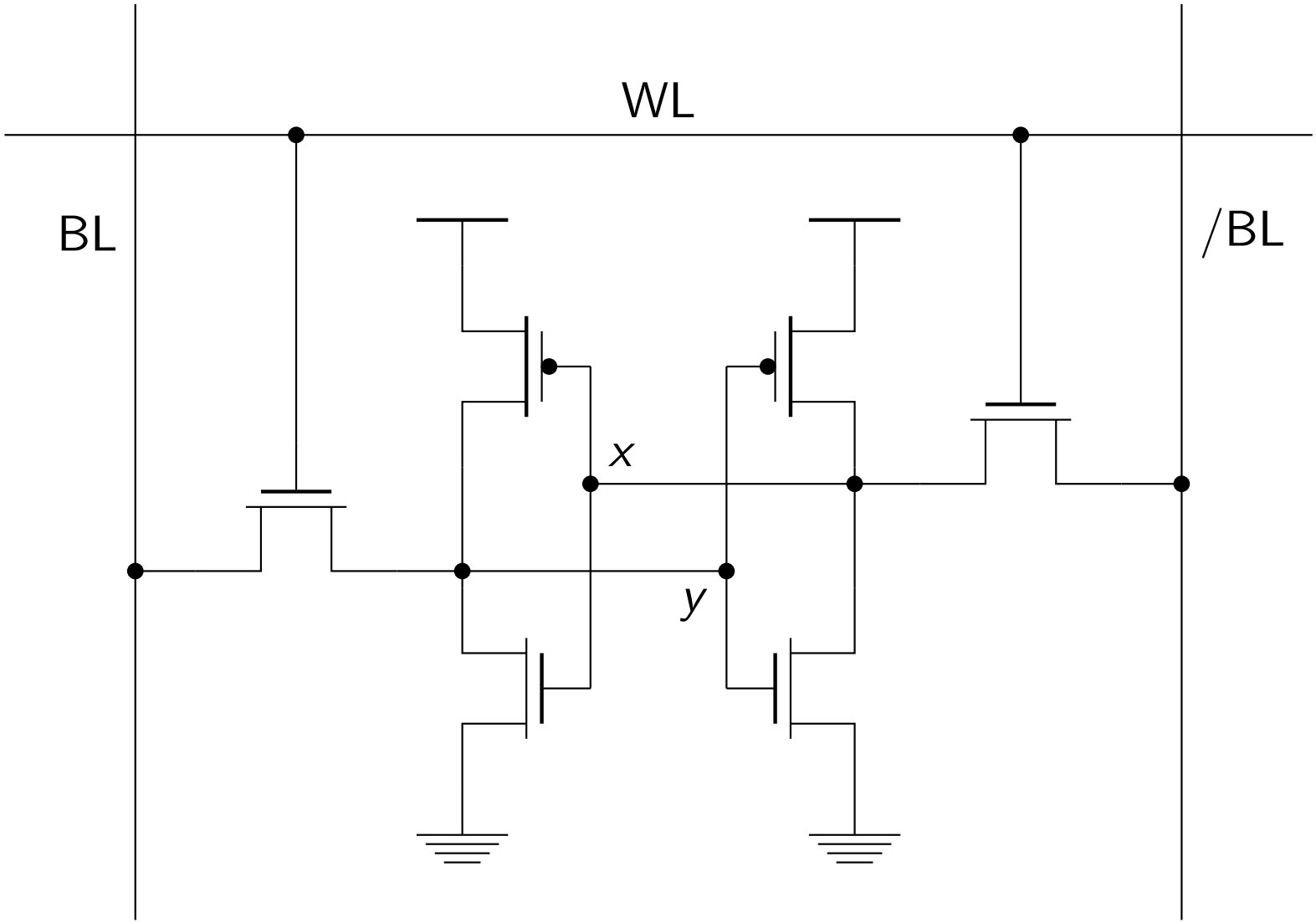
gnucap-uf, state space of a trigger



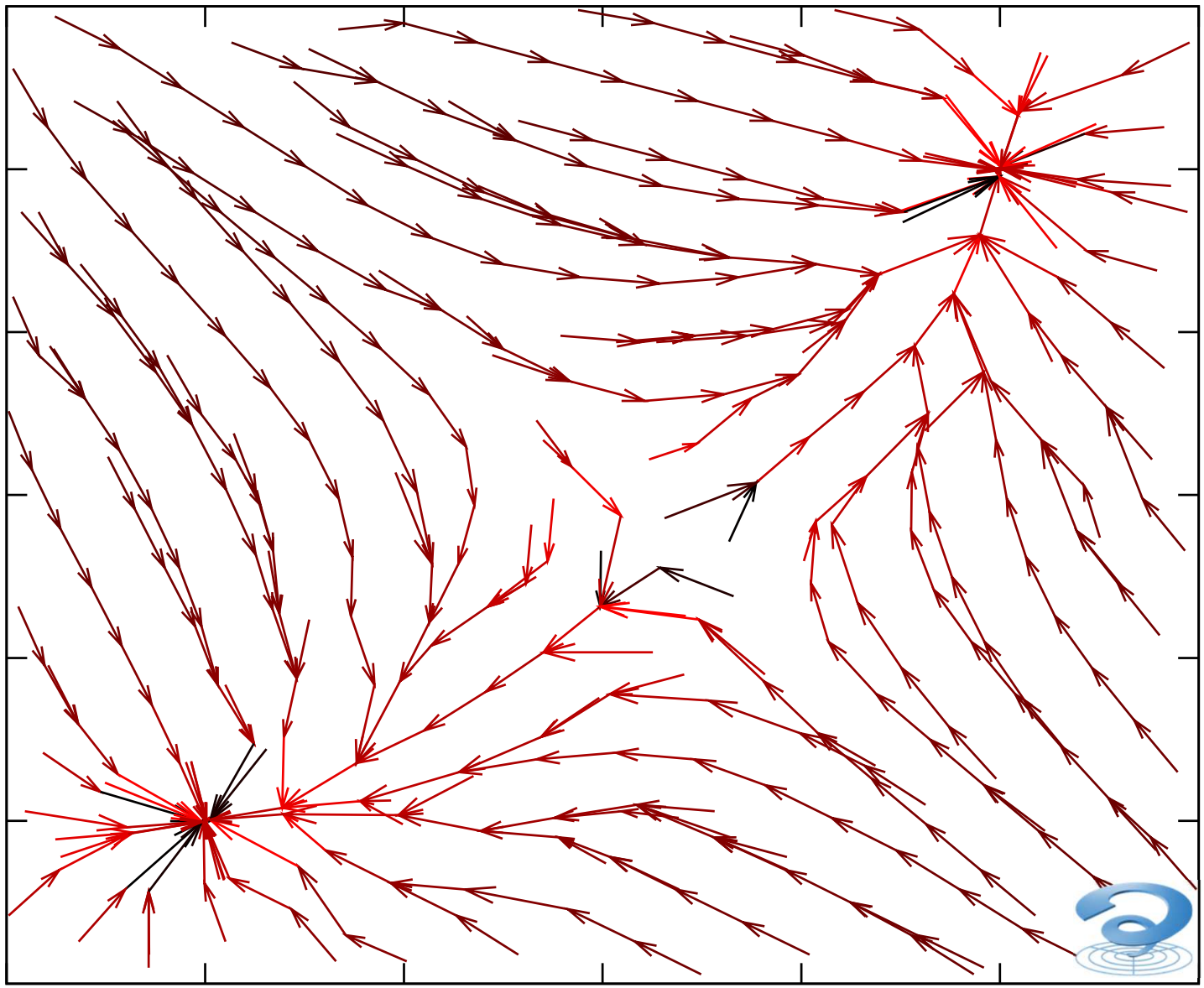
gnucap-uf, state space of a trigger



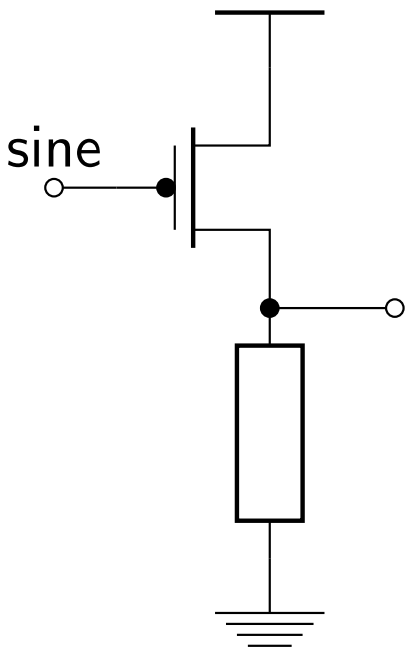
gnucap-uf sram cell discretization

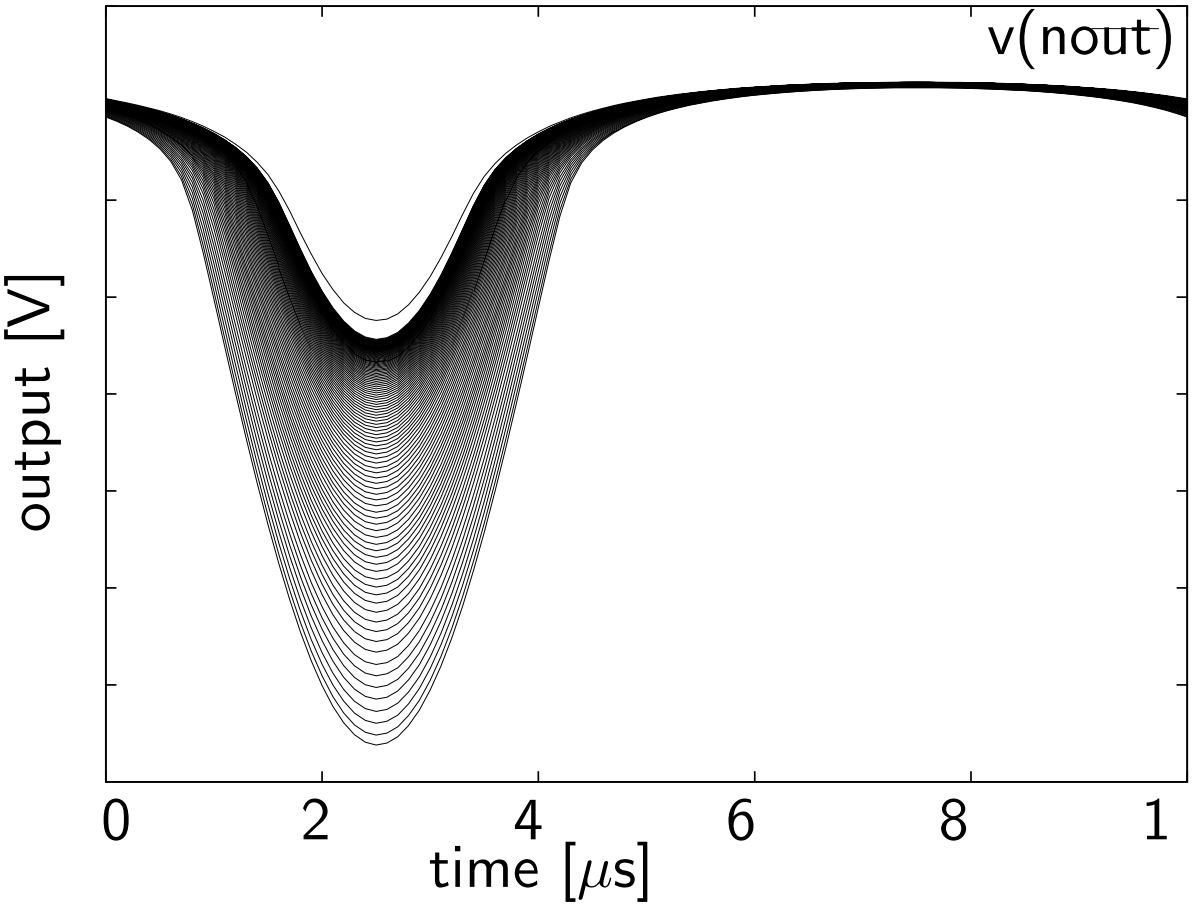


gnucap-uf sram cell discretization

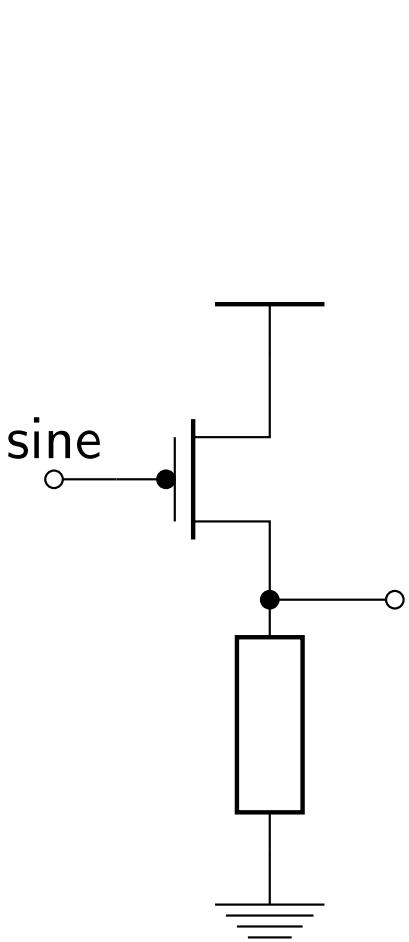


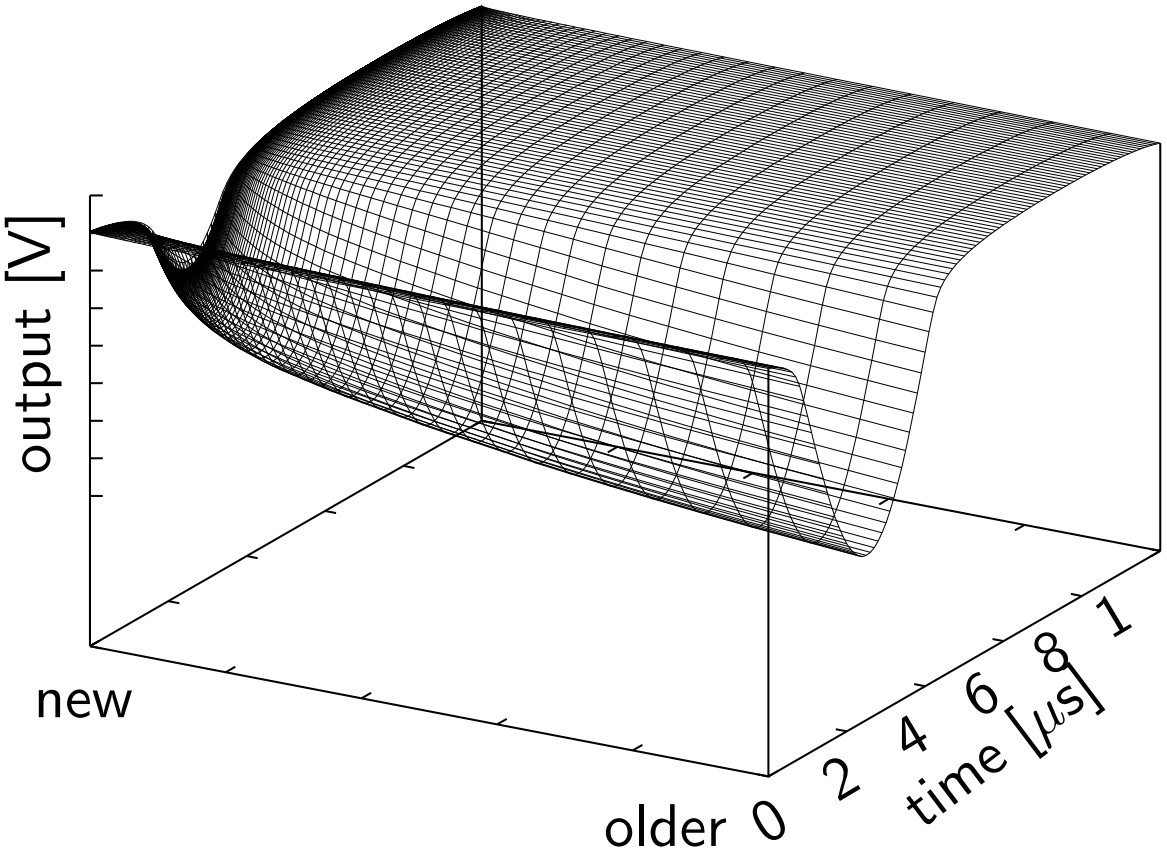
Ageing simulation with gnucap-uf





Ageing simulation with gnucap-uf





gnucap-adms

- Turns verilog-a models into component plugins
- Uses admsXml
- Based on a student project (A. Fröse)
  - Derived from gnucap-mot-adms (G. Serdyuk)
    - Derived from several mot-adms templates
    - .. including (ng?)spice adaptions

gnucap-adms

- Turns verilog-a models into component plugins
- Uses admsXml
- Based on a student project (A. Fröse)
  - Derived from gnucap-mot-adms (G. Serdyuk)
    - Derived from several mot-adms templates
    - .. including (ng?)spice adaptions
- Implementation goals/motivation (roughly)
  - (earlier) mixed signal simulation and compact modelling
  - Multiple disciplines simulation
  - Formal verification, 'equivalence' checking etc.
  - Circuit level ageing models and simulation

gnucap-adms today

- More standard support
  - Voltage sources, current probes
  - Verilog style disciplines

gnucap-adms today

- More standard support
  - Voltage sources, current probes
  - Verilog style disciplines
  - Subcircuit component instanciation
  - Linear operators (`idt`, `ddx`)

gnucap-adms today

- More standard support
  - Voltage sources, current probes
  - Verilog style disciplines
  - Subcircuit component instanciation
  - Linear operators (`idt`, `ddx`)
  - bsim6 works (partially)

gnucap-adms today

- More standard support
  - Voltage sources, current probes
  - Verilog style disciplines
  - Subcircuit component instanciation
  - Linear operators (`idt`, `ddx`)
  - bsim6 works (partially)
  - sensitivity, noise, switching branches, events, ...

gnucap-adms today

- More standard support
  - Voltage sources, current probes
  - Verilog style disciplines
  - Subcircuit component instanciation
  - Linear operators (`idt`, `ddx`)
  - bsim6 works (partially)
  - sensitivity, noise, switching branches, events, ...
- Refactoring in progress

gnucap-adms today

- More standard support
  - Voltage sources, current probes
  - Verilog style disciplines
  - Subcircuit component instanciation
  - Linear operators (`idt`, `ddx`)
  - bsim6 works (partially)
  - sensitivity, noise, switching branches, events, ...
- Refactoring in progress
  - Bug reports/fixes,
  - Feature additions,
  - Unit tests are welcome.

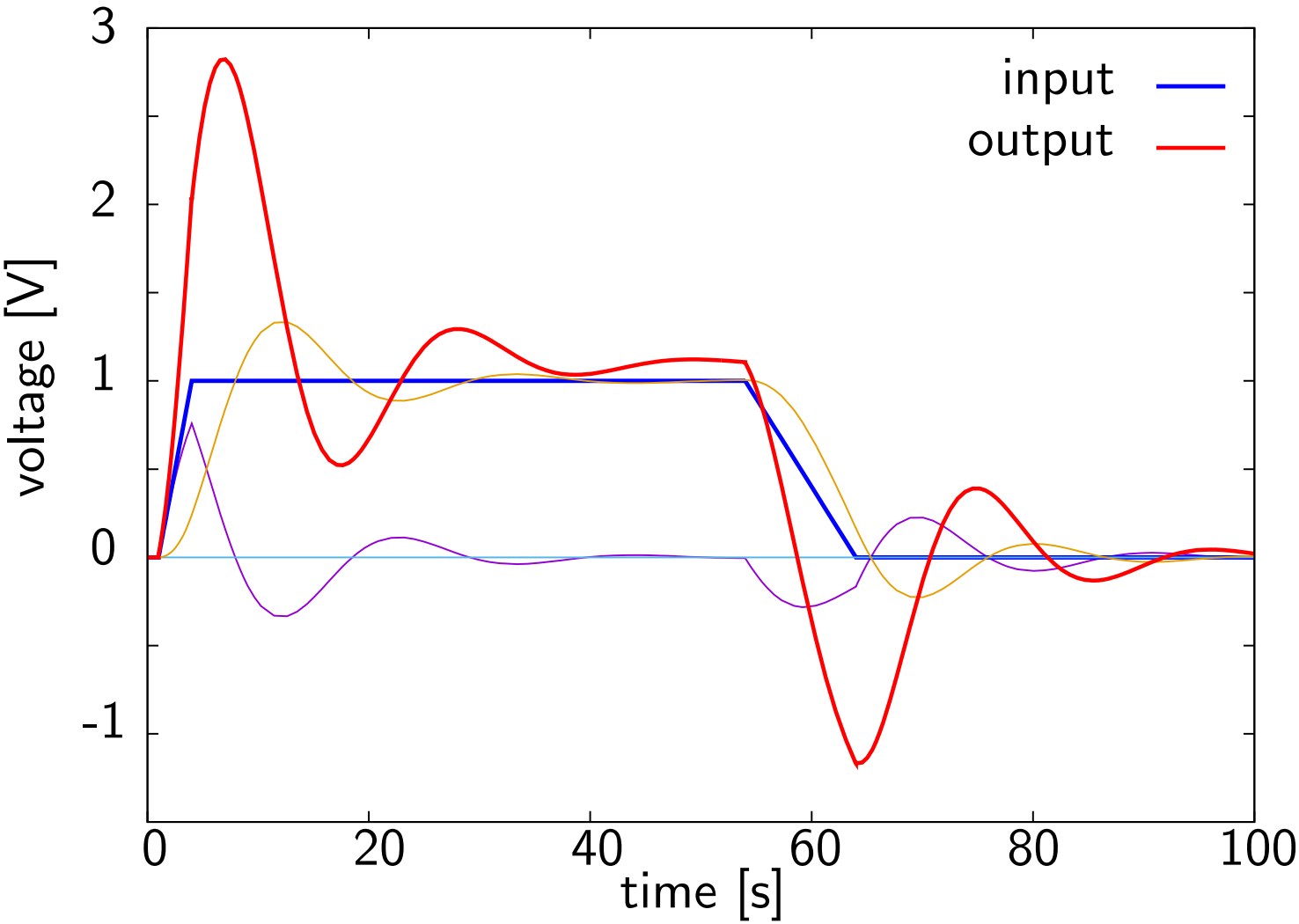
gnucap-adms inline example

```
load lang_adms.so
adms
`include "discipline.h"

module pid(sp,sn,cp,cn);
   in sp,sn,cp,cn;
   electrical sp,sn,cp,cn;
   parameter real p = 1 from [0:inf);
   parameter real i = 1 from [0:inf);
   parameter real d = 1 from [0:inf);
   analog begin
      V(sp,sn) <+ p * V(cp,cn);
      V(sp,sn) <+ i * idt(V(cp,cn));
      V(sp,sn) <+ d * ddt(V(cp,cn));
   end
endmodule
endadms

[..] instance, testbench, sim command
```

gnucap-adms inline example



gnucap-adms ageing extension

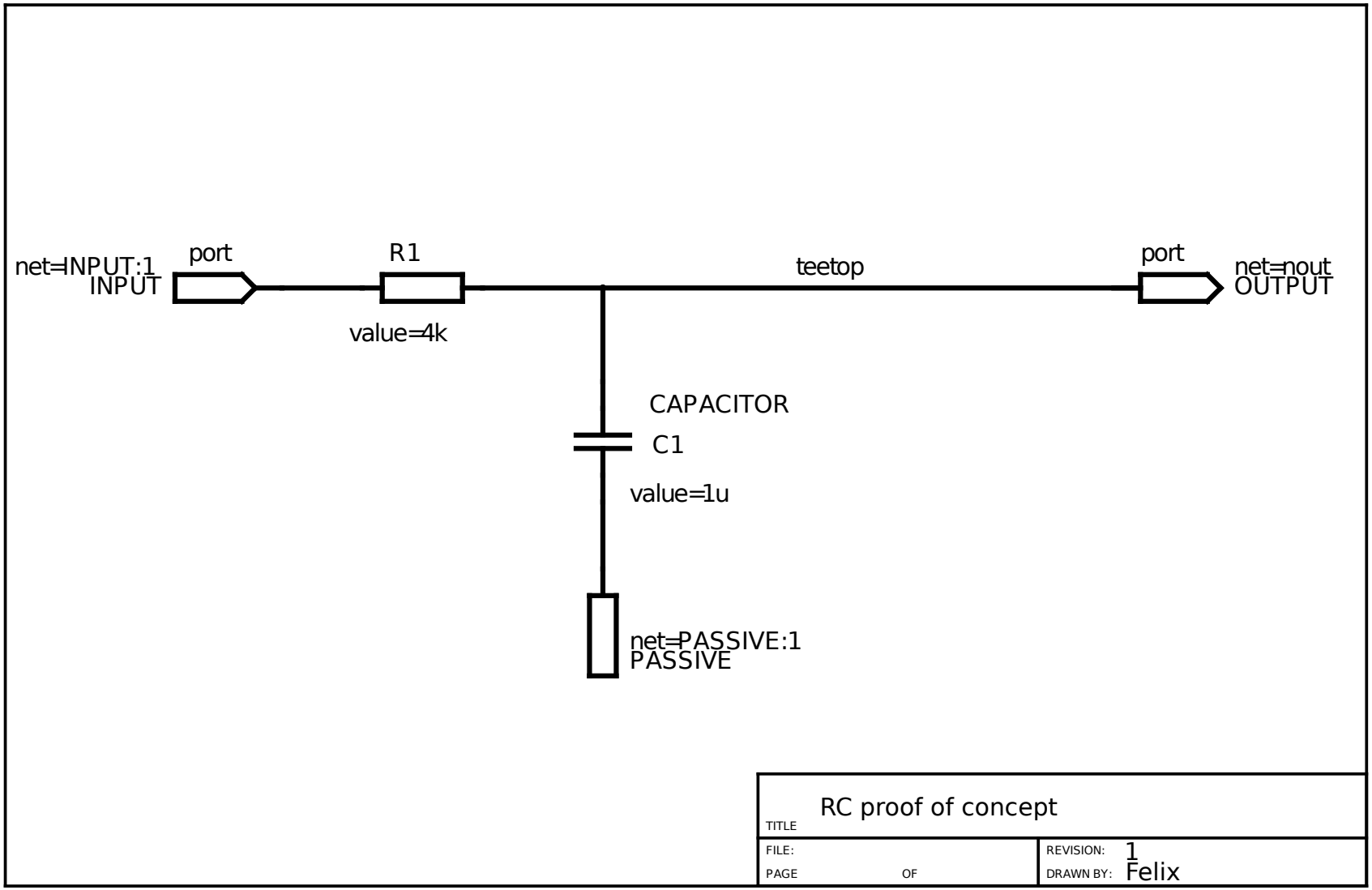
```
module ageing_component(a, b, c);
   electrical a, b, c;
   degradational d0, .., dk;
   [ parameters, variables, functions .. ]
   ageing_process_subdevice_0 AP0(d0); ..
   ageing_process_subdevice_k APk(dk);
   analog begin
     param = f_p(State(d0),
           + [..] + State(dk));
     I(a,b) <+ f_bm(param,V(a,b),V(b,c));
     Level(d0) <+ f_L0(V(a), V(b), V(c));
     [..]
     Level(dk) <+ f_Lk(V(a), V(b), V(c));
   end
endmodule
```

gnucap-geda

- originally a GSoC project (2012?, Savant Krishna)
  - idea: schematic representation using generic component patterns
  - implementation: gEDA schematic parser (gnucap plugin)
- convert schematic (nets, symbols) to any format
- convert anything to gEDA schematics (needs work)
- analyse/simulate (hierarchical) schematic + component library
- provide component library supplementing gEDA symbols (stub)

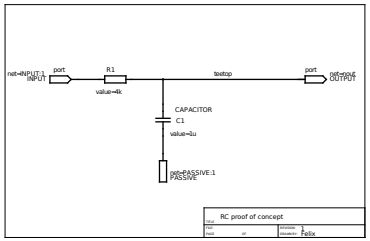
gnucap-geda, example

- Simple geda schematic (`rc.sch`)



gnucap-geda, example

- Simple geda schematic (`rc.sch`)

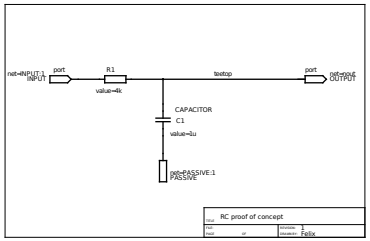


- Verilog netlist representation

```
module rc.sch (INPUT:1,nout,PASSIVE:1);
   //v 20130925 2
   net #() net0 (.p(x_nn_0),.n(x_nn_1));
   place #(.x(47300),.y(46600)) 47300:46600 (.port(x_nn_0));
   place #(.x(47300),.y(47300)) 47300:47300 (.port(x_nn_1));
   port #(.basename(input-2.sym),.net(INPUT:1)) INPUT:1 (.int(x_cn_2),.ext(INPUT:1));
   place #(.x(44900),.y(47300)) 44900:47300 (.port(x_cn_2));
   port #(.basename(output-2.sym),.net(nout)) nout (.int(x_cn_3),.ext(nout));
   place #(.x(53500),.y(47300)) 53500:47300 (.port(x_cn_3));
   RESISTOR #(.basename(resistor-2.sym),.value(4k)) R1 (.1(x_cn_4),.2(x_cn_5));
   place #(.x(45500),.y(47300)) 45500:47300 (.port(x_cn_4));
   place #(.x(46400),.y(47300)) 46400:47300 (.port(x_cn_5));
   net #() net1 (.p(x_cn_4),.n(x_cn_2));
   net #() net2 (.p(x_nn_6),.n(x_nn_7));
   place #(.x(47300),.y(45200)) 47300:45200 (.port(x_nn_6));
   place #(.x(47300),.y(45700)) 47300:45700 (.port(x_nn_7));
   port #(.basename(passive-1.sym),.net(PASSIVE:1)) PASSIVE:1 (.int(x_nn_6),.ext(PASSIVE:1));
   CAPACITOR #(.basename(capacitor-1.sym),.description(capacitor),
               .numslots(0),.symversion(0.1),.value(1u)) C1 (.1(x_nn_7),.2(x_nn_0));
   net #() teetop (.p(x_cn_5),.n(x_cn_3));
   net #() extranet4 (.p(x_cn_5),.n(x_nn_1));
endmodule // rc.sch
```

gnucap-geda, example

- Simple geda schematic (`rc.sch`)



- Verilog netlist representation

```
module rc.sch (INPUT:1,nout,PASSIVE:1);
   net #() net0 (.p(x_nn_0),.n(x_nn_1));
   place #(.x(47300),.y(46600)) 47300:46600 (.port(x_nn_0));
   RESISTOR #(.basename(resistor-2.sym),.value(4k)) R1 (.1(x_cn_4),.2(x_cn_5));
   [..]
endmodule;
```

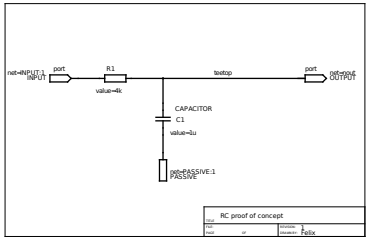
- Ready for simulation

```
geda rc.sch module
verilog
rc.sch sch1(n1 n2 n0);
spice
V1 n1 n0 pulse iv=0 pv=1 delay=0 width=10m
R1 n0 0 1

.print tran v(nodes)
.tran 0 20m .1m basic trace=n
```

gnucap-geda, example

- Simple geda schematic (`rc.sch`)

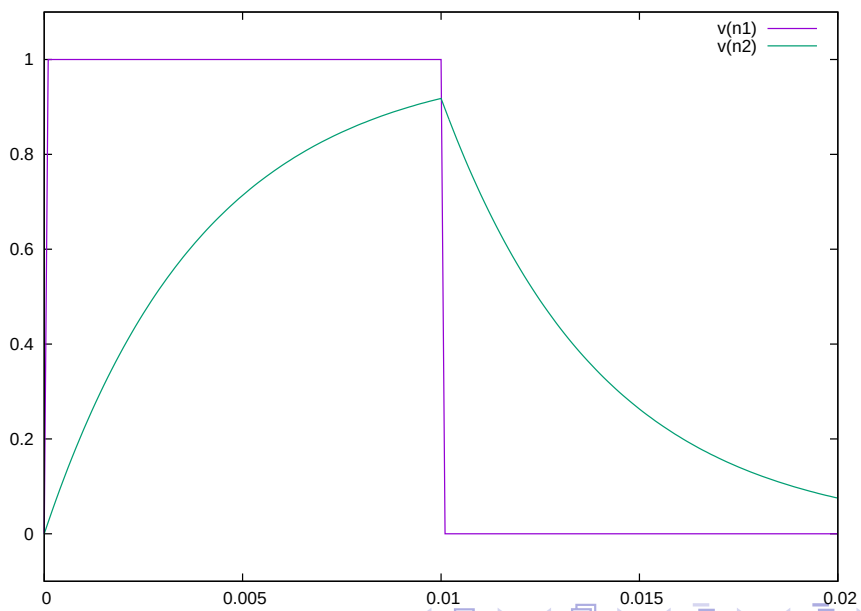


- Verilog netlist representation

```
module rc.sch (INPUT:1,nout,PASSIVE:1);
   net #() net0 (.p(x_nn_0),.n(x_nn_1));
   place #(.x(47300),.y(46600)) 47300:46600 (.port(x_nn_0));
   RESISTOR #(.basename(resistor-2.sym),.value(4k)) R1 (.1(x_cn_4),.2(x_cn_5));
   [..]
endmodule;
```

- Ready for simulation

```
geda rc.sch module
.spice
[..]
.print tran v(nodes)
.tran 0 20m .1m basic trace=n
```

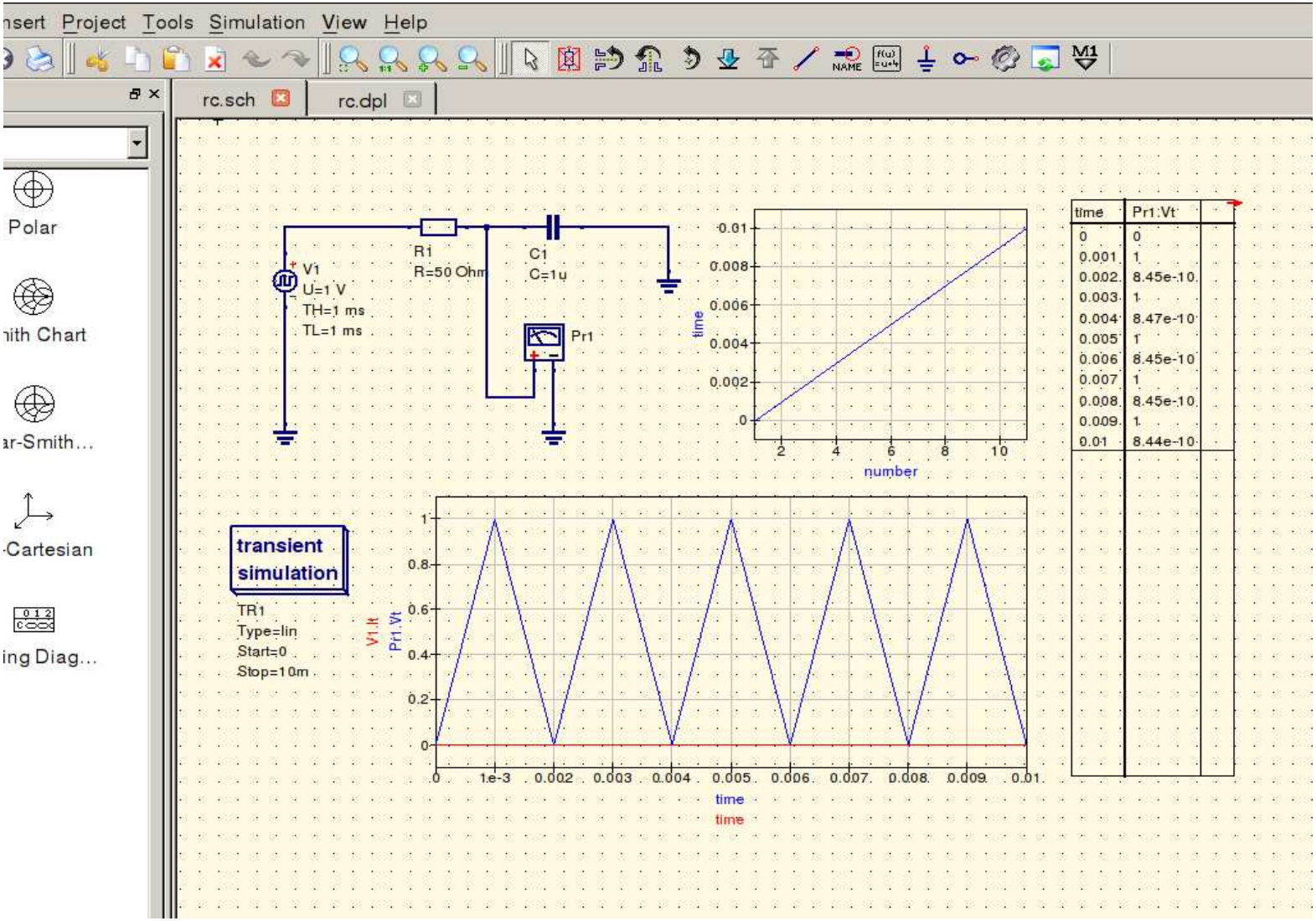


gnucap-qucs

- Proof of concept initiated after FOSDEM 2015
  (based on old code fragments by Fabian Vallon)
- a *qucsator* replacement by means of
  - Input deck parser (some SPICE/ADS inspired format)
  - Compatible components (names, pins, parameters...)
  - Emulate commands and semantics (noninteractive, one-shot, probe-placement)
  - Turn results into "qucs dataset" format ('`.dat`')
  - Infinite possibilities.

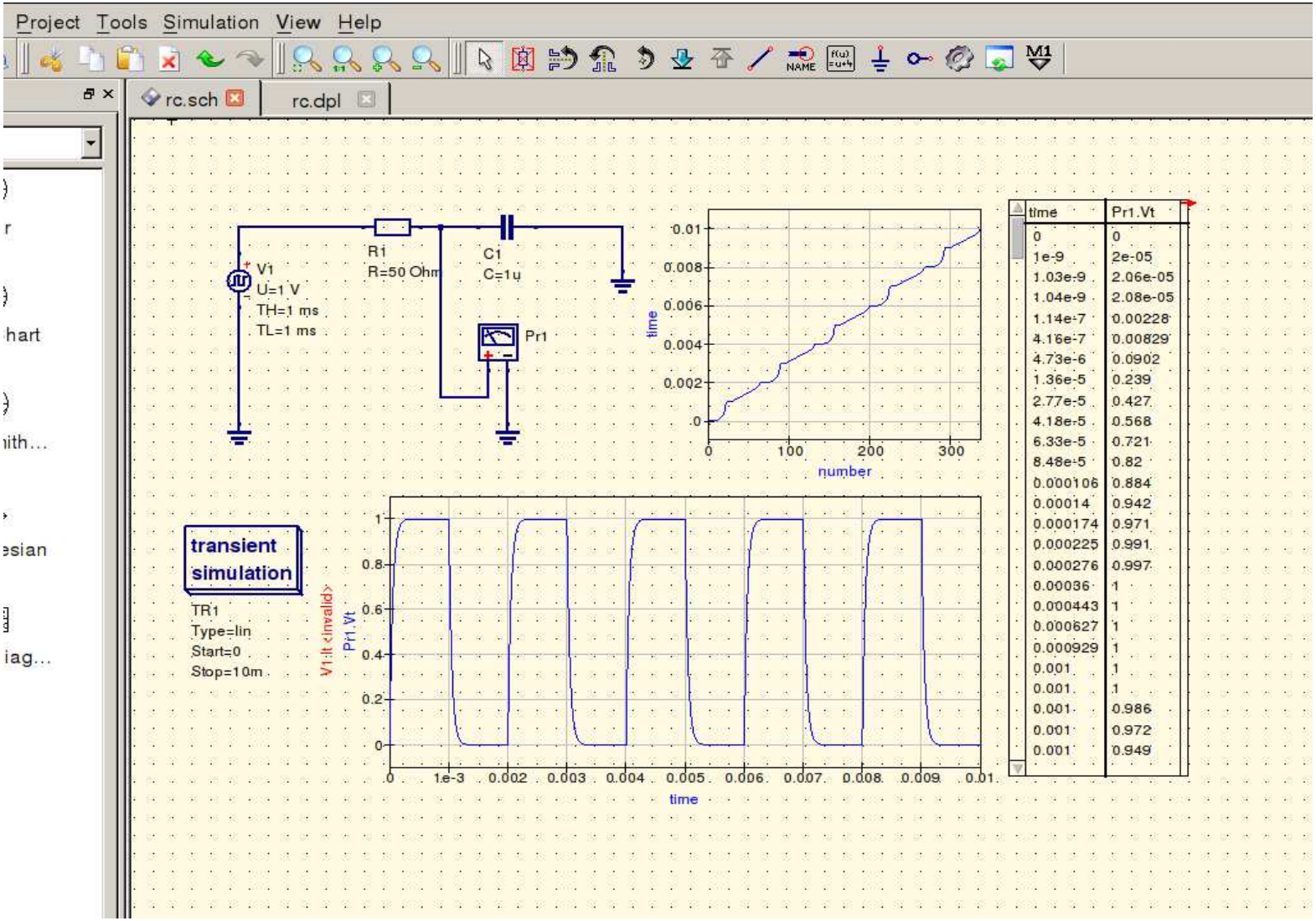
gnucap-qucs demo

```
$ QUCSATOR=qucsator qucs -i rc.sch
```



gnucap-qucs demo

```
$ QUCSATOR=gnucsator.sh qucs -i rc.sch
```



gnucap-qucs roadmap

- What we have

gnucap-qucs roadmap

- What we have
  - few components (just wrapped)
  - transient command wrapper
  - intermediate output (plus suboptimal conversion)
  - parameters are incomplete

gnucap-qucs roadmap

- What we have
  - few components (just wrapped)
  - transient command wrapper
  - intermediate output (plus suboptimal conversion)
  - parameters are incomplete
  - works with unmodified QUCS (simple patch already merged)
  - should work with unmodified gnucap

gnucap-qucs roadmap

- What we have
  - few components (just wrapped)
  - transient command wrapper
  - intermediate output (plus suboptimal conversion)
  - parameters are incomplete
  - works with unmodified QUCS (simple patch already merged)
  - should work with unmodified gnucap
- Proper integration: more work (on all ends)
  - Output plugins (hdf? shm?)
  - More modular QUCS

gnucap-qucs roadmap

- What we have
  - few components (just wrapped)
  - transient command wrapper
  - intermediate output (plus suboptimal conversion)
  - parameters are incomplete
  - works with unmodified QUCS (simple patch already merged)
  - should work with unmodified gnucap
- Proper integration: more work (on all ends)
  - Output plugins (hdf? shm?)
  - More modular QUCS
  - Needs your help.

Thank You.