MAPP:
The Berkeley Model and Algorithm Prototyping Platform

**Tianshi Wang, Karthik Aadithya, Bichen Wu,
Jian Yao and Jaijeet Roychowdhury**

EECS Department
University of California at Berkeley

Motivation for MAPP

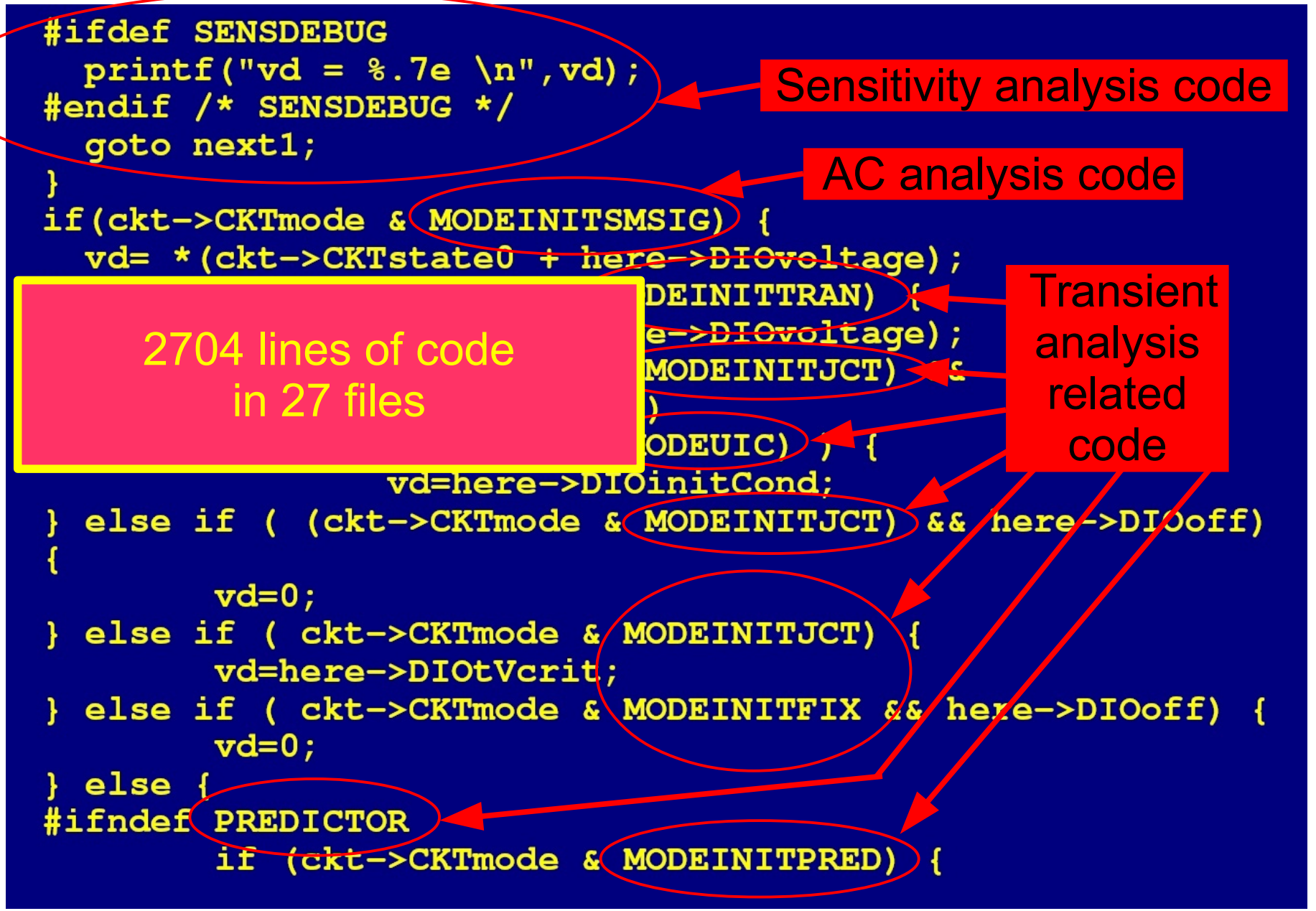
- **Berkeley Model and Algorithm Prototyping Platform**
- Developing **good** compact models: many pitfalls
  - » examples: discontinuities/smoothness, well-posedness
    - – problems usually discovered at deployment (ie, during simulation)
  - » problems often hard to debug and resolve
    - – compact model developer and simulator people blame each other
- Anyone working the area of simulation today needs
  - » device models: basic electrical elements, semiconductor devices, …
  - » base algorithms: **robust nonlinear solution**, transient, HB/shooting, …
  - » parsing, equation formulation, output, …
  - » huge (waste of) effort of re-development of basic capabilities
- Goal of MAPP: to ease these problems
  - » common, open-source simulation framework
  - » in MATLAB
    - – empowers non-programmers to debug models and algorithms

Why not use SPICE?

- SPICE: the original open-source simulator
  - » de-facto standard
  - » structure (SPICE3): all analyses in all models
  - » prototyping models & algorithms: takes months to years
    - –e.g., shooting method (S-SPICE)

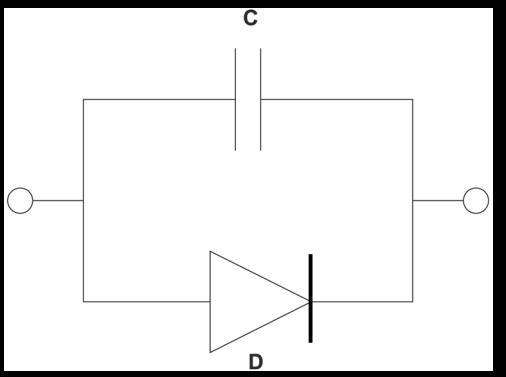
- To be useful: **modular, well-structured, flexible**
  - » separated models, algorithms, numerics, I/Os
  - » simple, clean interfaces
  - » short, easy to read, easy to modify

Excerpt from *dioload.c* (SPICE3)



Glimpse: Diode Model in MAPP

```
 1 function MOD = diodeCapacitor_ModSpec_wrapper()
 2 % ModSpec description of an ideal diode in parallel with a capacitor
 3     MOD = ee_model();
 4     MOD = add_to_ee_model(MOD, 'external_nodes', {'p', 'n'});
 5     MOD = add_to_ee_model(MOD, 'explicit_outs', {'ipn'});
 6     MOD = add_to_ee_model(MOD, 'parms', {'C',2e-12, 'Is',1e-12, 'VT',0.025});
 7     MOD = add_to_ee_model(MOD, 'f', @f);
 8     MOD = add_to_ee_model(MOD, 'q', @q);
 9 end
10
11 function out = f(S)
12     v2struct(S);
13     out = Is*(exp(vpn/VT)-1);
14 end
15
16 function out = q(S)
17     v2struct(S);
18     out = C*vpn;
19 end
"diodeCapacitor_ModSpec_wrapper.m" 19L, 548C written          1,1           All
```

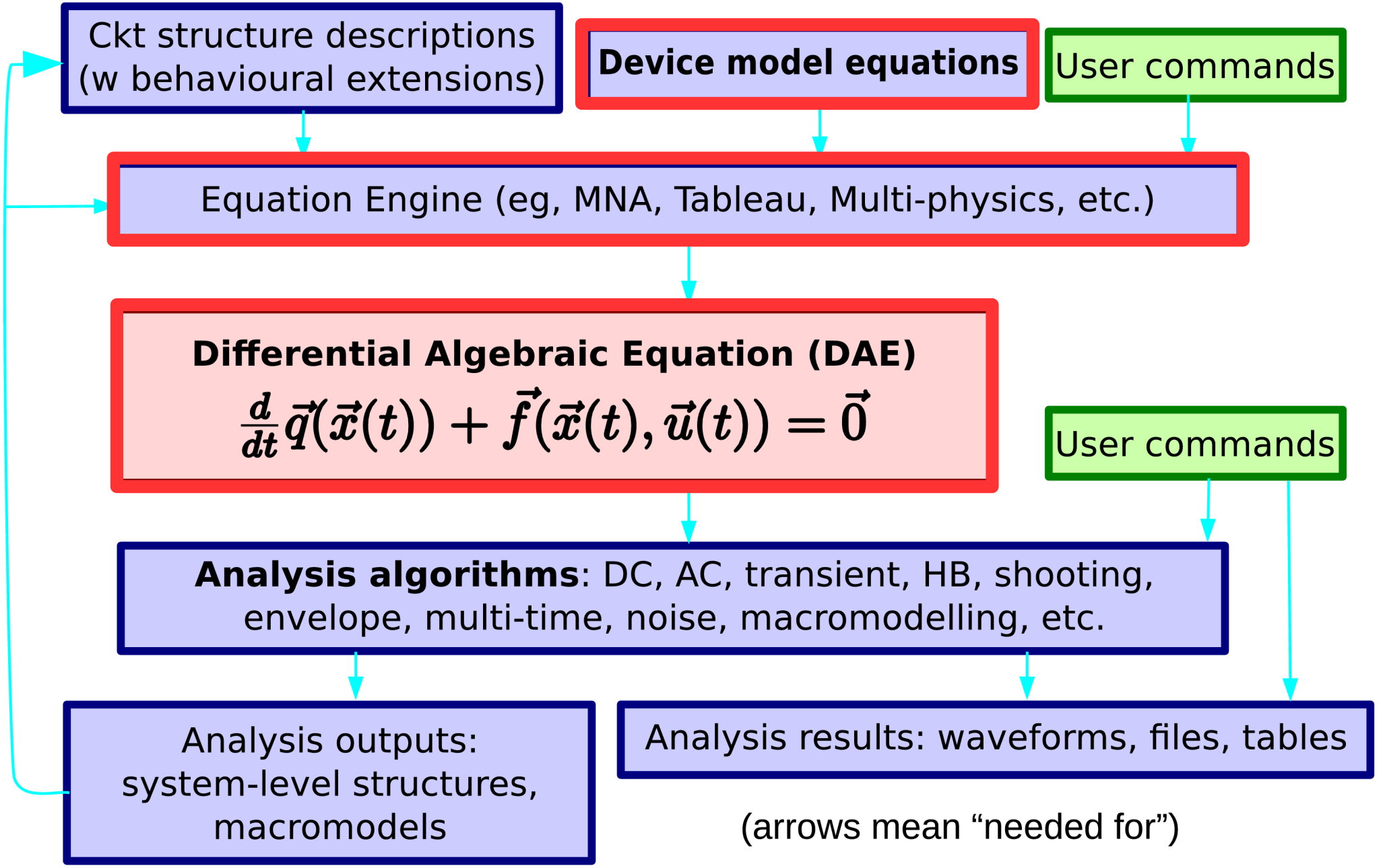


MOD.terminals
MOD.parms
MOD.explicit_outs
MOD.f: function handle
MOD.q: function handle
…

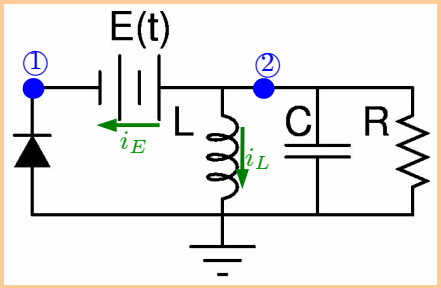
MODSPEC format

- **executable (in Matlab)**
- **takes 10min to write**
- **works in all analyses**

Code Structuring of MAPP



Differential Algebraic Equations (DAEs)



$$\text{KCL}_1: \quad -\text{diode}(-e_1; I_S, V_t) - i_E = 0$$

$$\text{KCL}_2: \quad i_E + i_L + \frac{e_2}{R} + C\frac{de_2}{dt} = 0$$

$$\text{BCR}_E: \quad e_2 - e_1 - E(t) = 0$$

$$\text{BCR}_L: \quad e_2 - L\frac{di_L}{dt} = 0$$

EE

state (unknown) vector
size n

input coupling matrix
size n x n_{inp}

inputs
size n_{inp}

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t)) + B\vec{u}(t) = \vec{0}$$

$$\vec{y}(t) = C\vec{x}(t) + D\vec{u}(t)$$

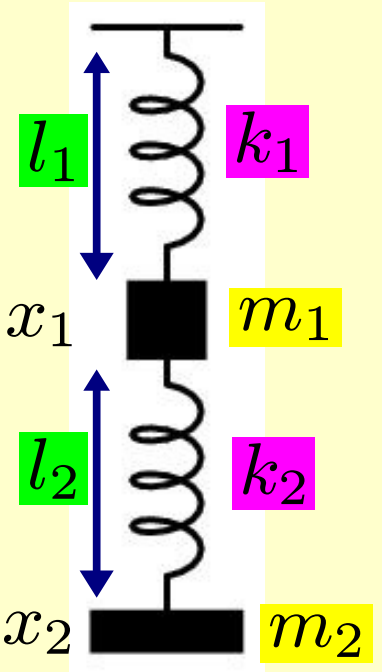
outputs
size n_{out}

state->output coupling matrix
size n_{out} x n

input->output feedthrough matrix
size n_{out} x n_{inp}

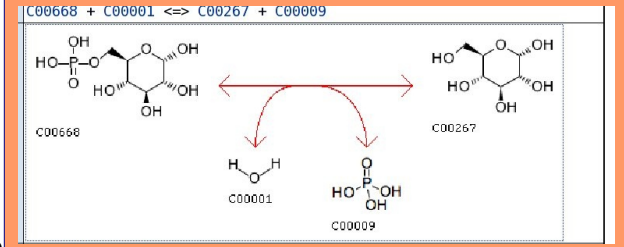
DAE
size n

Mechanical



$$m_1\dot{y}_1 = -k_1(x_1 - l_1) + k_2(x_2 - x_1 - l_2)$$
$$m_2\dot{y}_2 = -k_2(x_2 - x_1 - l_2)$$
$$\dot{x}_1 = y_1$$
$$\dot{x}_2 = y_2$$

Bio-Chemical

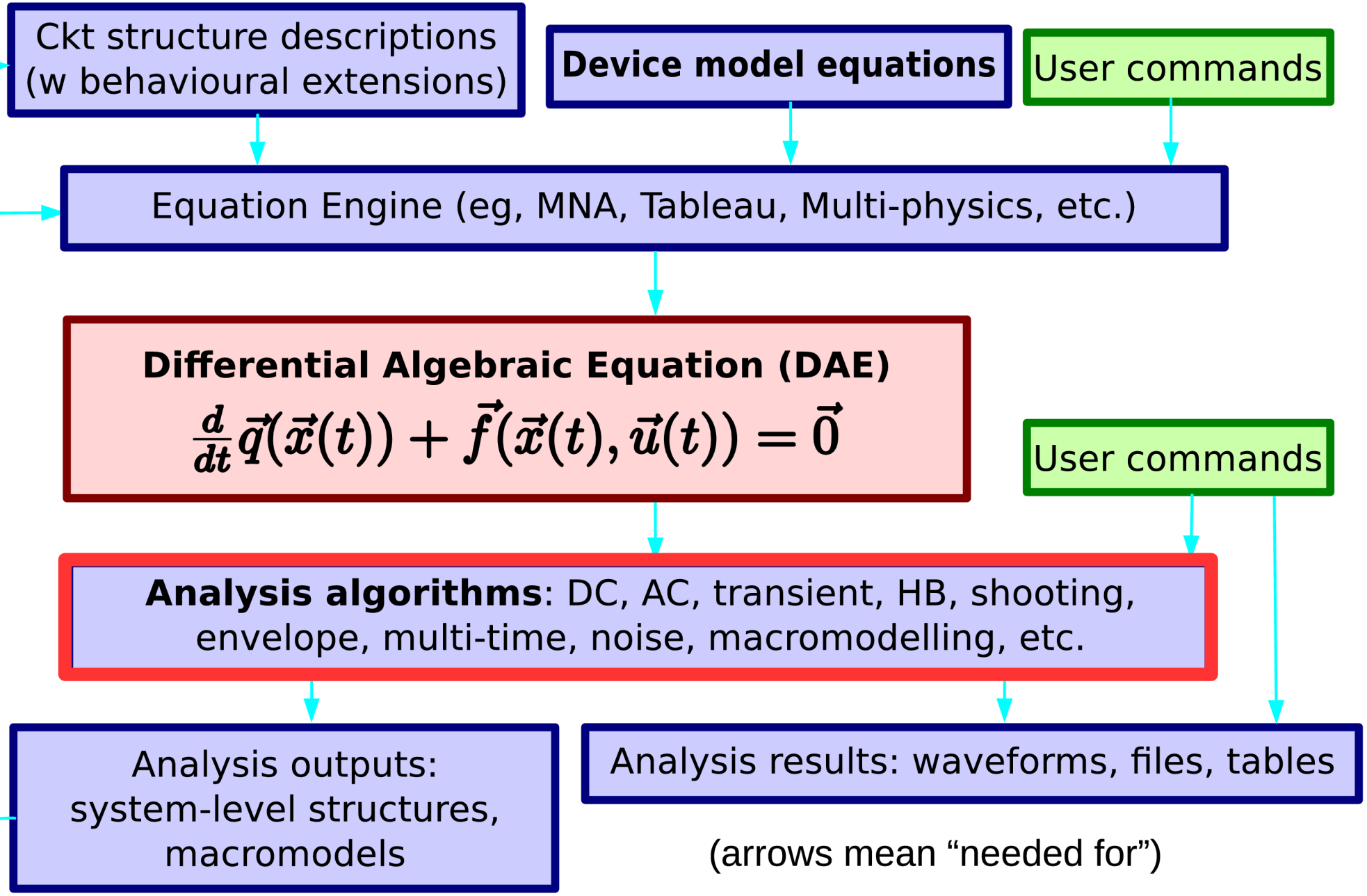


$$A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} C$$

$$C + D \underset{k_4}{\overset{k_3}{\rightleftharpoons}} E + A$$

$$R_1 \triangleq k_1[A][B] - k_2[C]$$
$$R_2 \triangleq k_3[C][D] - k_4[E][A]$$
$$\frac{d[A]}{dt} = -R_1 + R_2$$
$$\frac{d[B]}{dt} = -R_1$$
$$\frac{d[C]}{dt} = R_1 - R_2$$
$$\frac{d[D]}{dt} = -R_2$$
$$\frac{d[E]}{dt} = R_2$$

MAPP: Analysis Algorithms



Glimpse: Shooting Method in MAPP

Shooting Algorithm in MAPP (pseudo-code)

```
shootObj = shoot(DAE): // constructor
 1: shootObj.DAE = DAE;
 2: shootObj.tranObj = LMS(DAE); // transient simulation object
 3: set up member functions: .solve, .g, and .J
 4: return  shootObj;

shootObj.solve (initguess, T):
 1: x0 ← NR(@g, @J, initguess);
 2: shootSols = tranObj.solve(x0, 0, T);
 3: return  shootSols;

shootObj.g (x0):
 1: tranSols = tranObj.solve(x0, 0, T);
 2: return  gout = tranSols(:, n) - x0;

shootObj.J (x0):
 1: tranSols = tranObj.solve(x0, 0, T);
 2: Ci_pre = DAE.dq_dx(x0);
 3: M = eye(n);
 4: for i = 2:n do
 5:     x = tranSols(:, i); u = inputs(:, i);
 6:     Ci = DAE.dq_dx(x); Gi = DAE.df_dx(x, u);
 7:     M = (Ci + (tpts(i) - tpts(i-1)) * Gi) \ Ci_pre * M;
 8:     Ci_pre = Ci;
 9: end for
10: return  Jout = M - eye(n);
```

150 lines of code

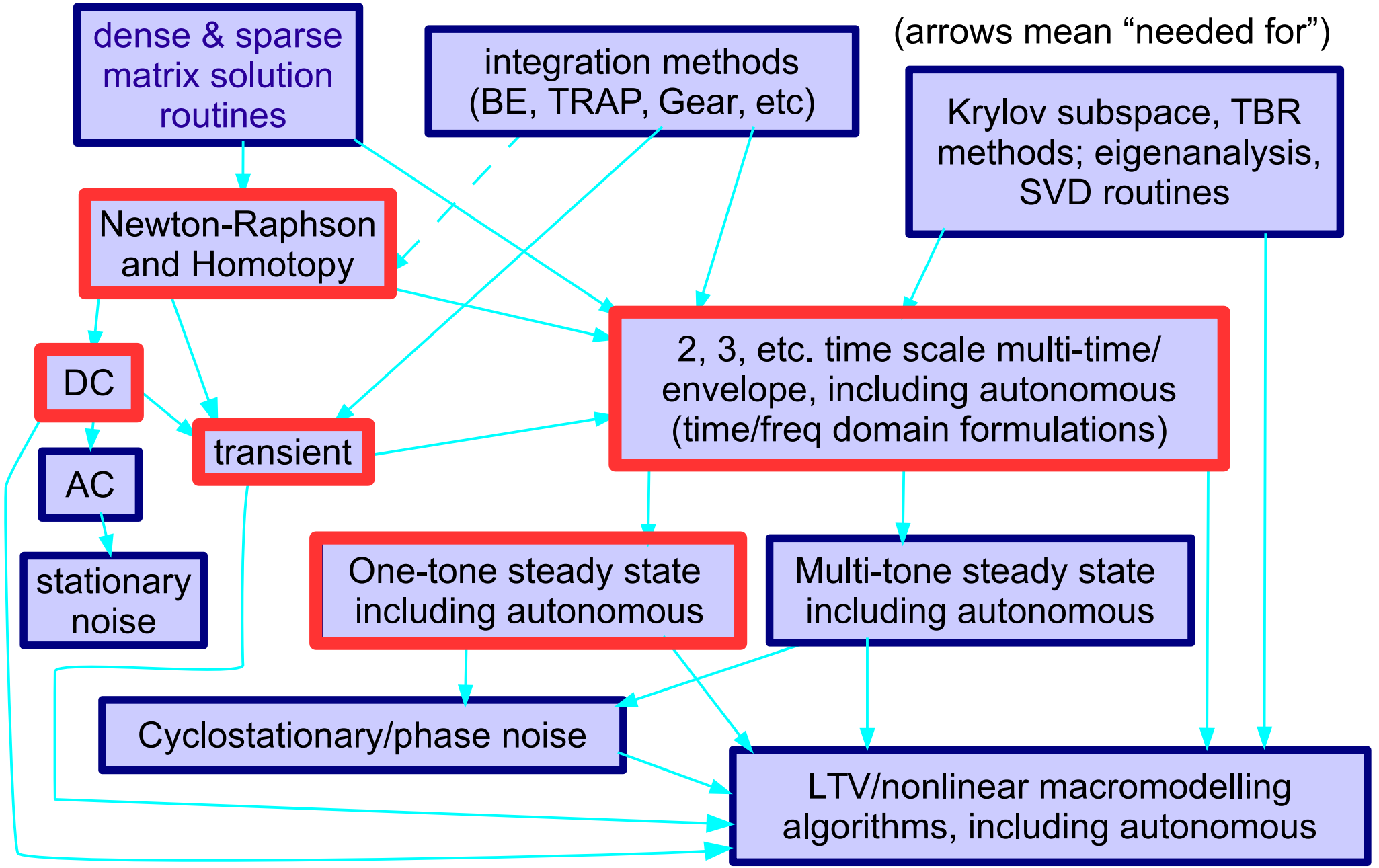
object-oriented

reuses LMS (transient) code

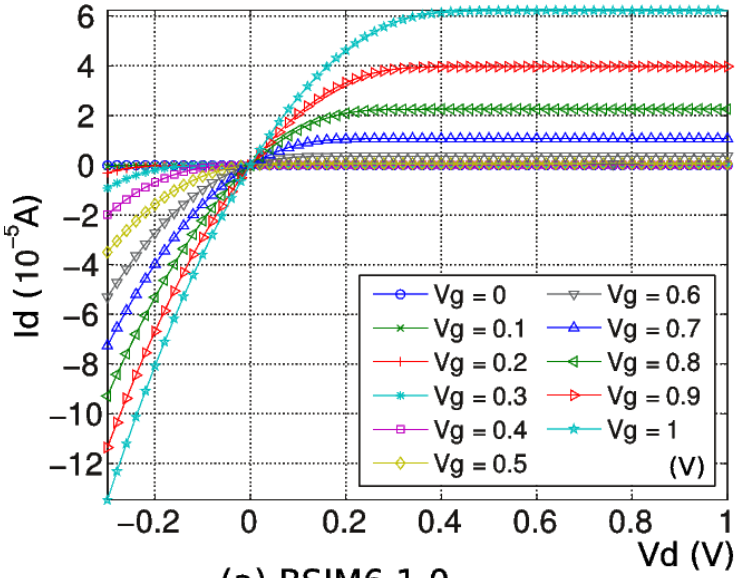
works with all devices, circuits, domains

code easy to write

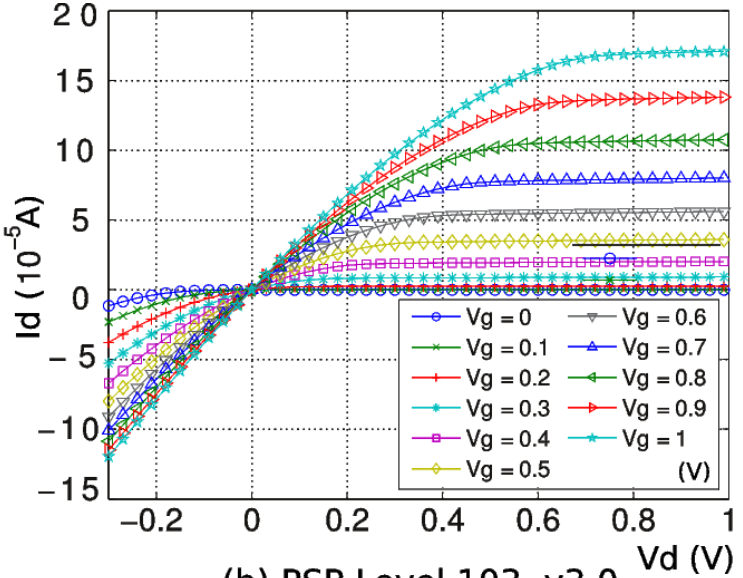
Analysis Algorithms: Structure



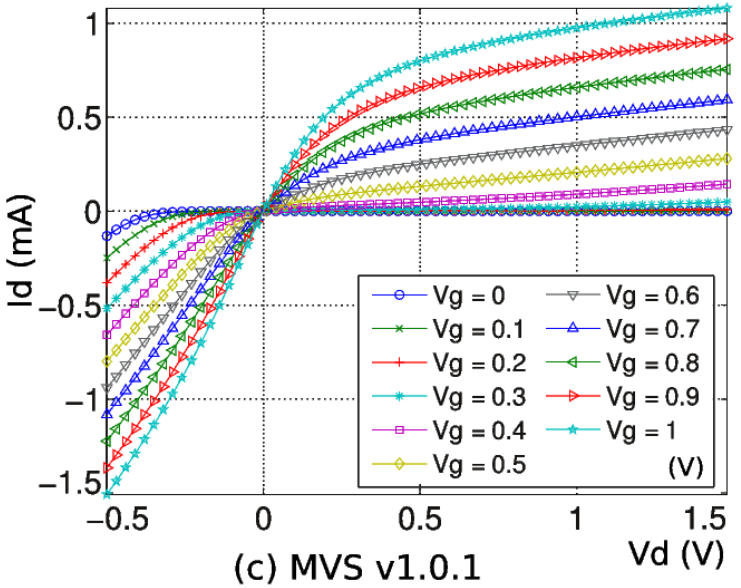
MAPP: Compact Model Prototyping



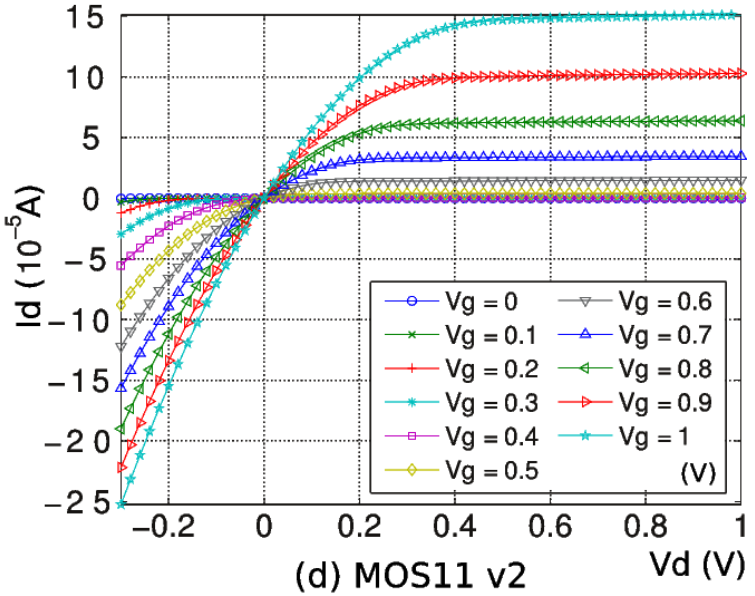
(a) BSIM6.1.0
default: L=10μm, W=10 μm



(b) PSP Level 103 v3.0
default: L=10μm, W=10 μm

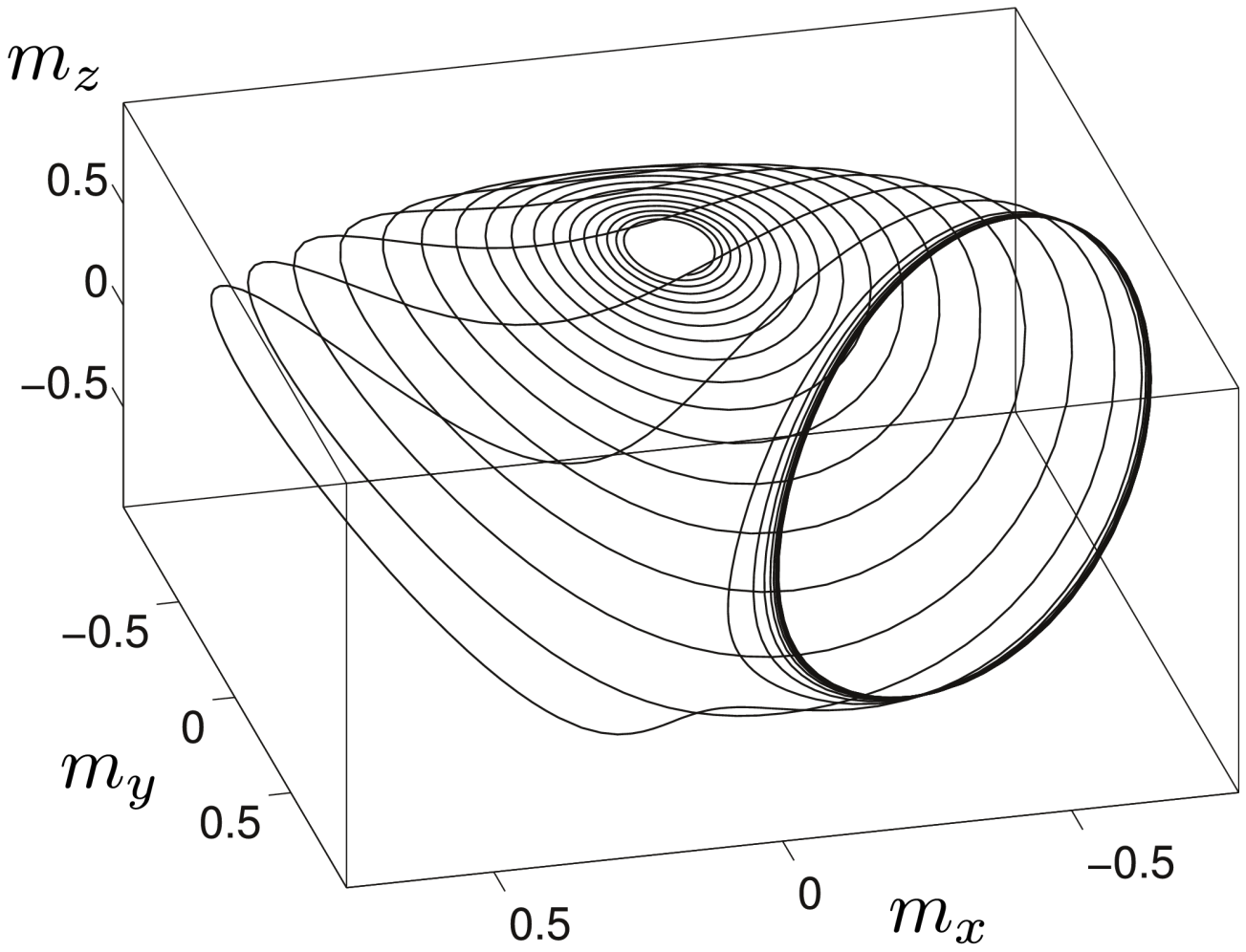


(c) MVS v1.0.1
default: L=80nm , W=1μm



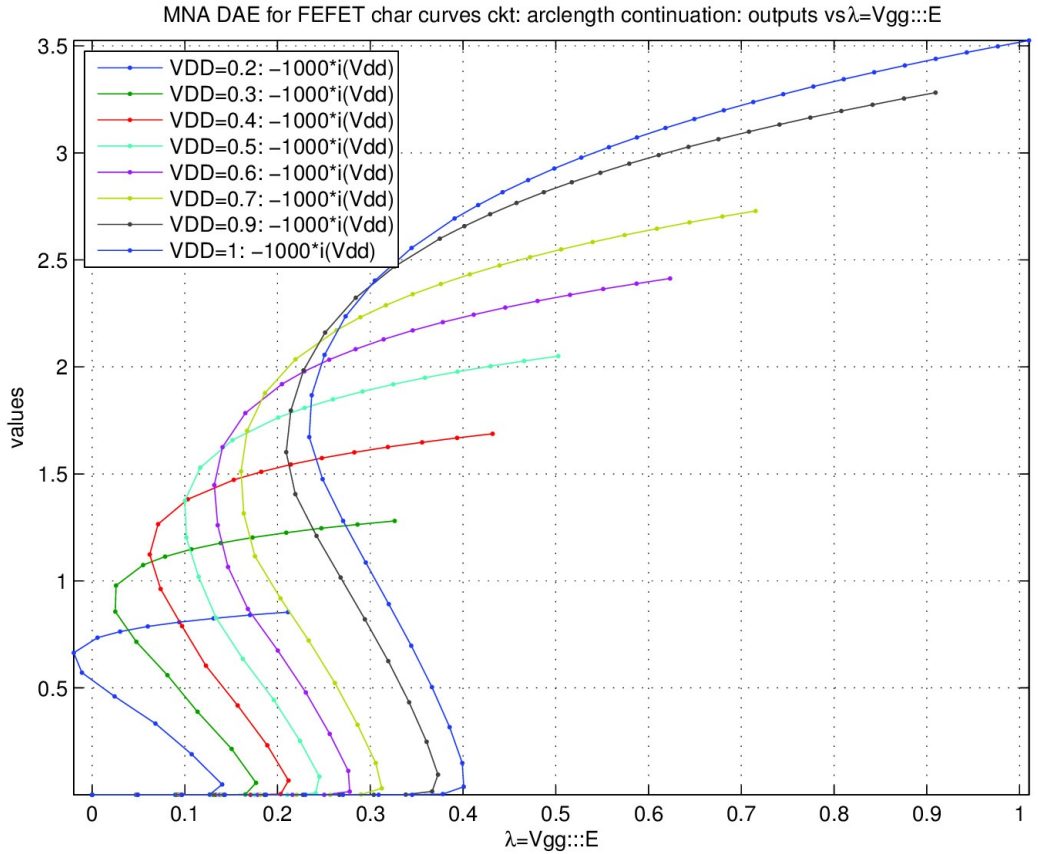
(d) MOS11 v2
default: L=1μm, W=1μm

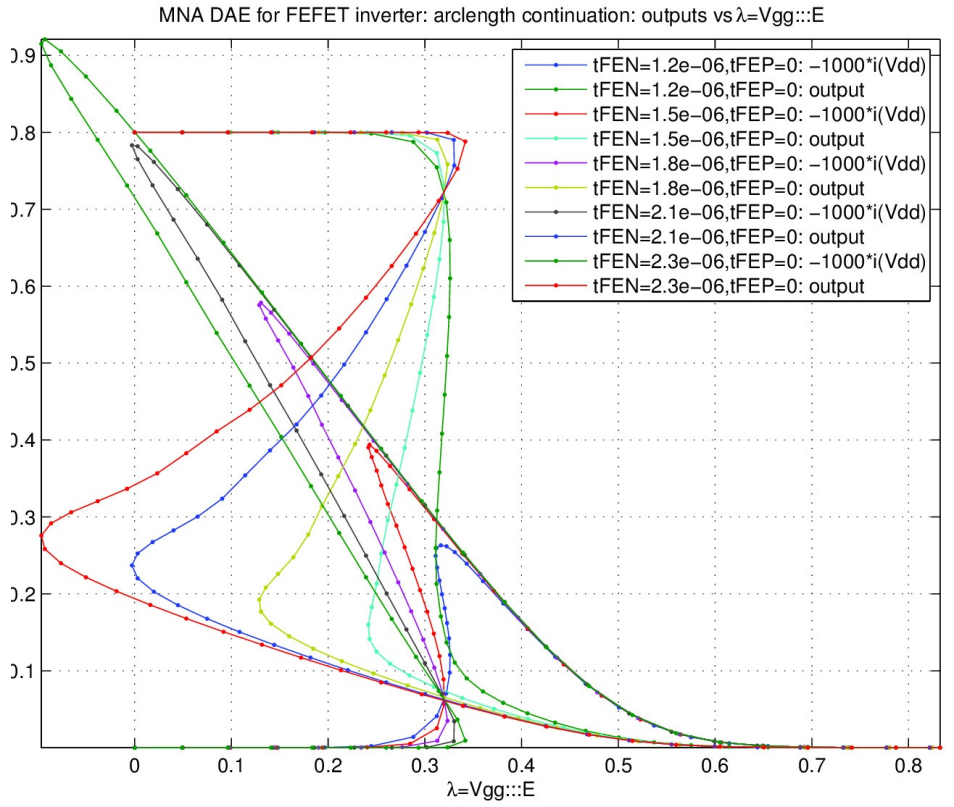
Landau-Lifshitz-Gilbert (LLG) Model: Spin-Torque Devices



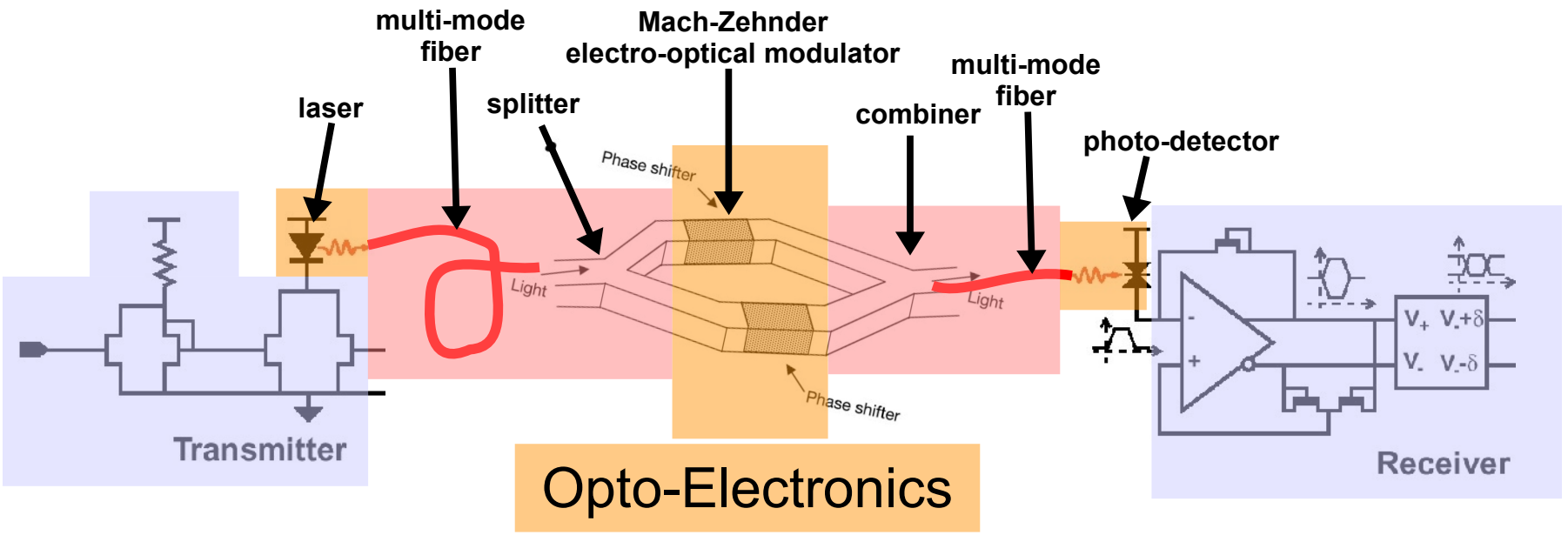
FEFET: Char Curves, Inverter

- Ferro-electrical FET (negative cap. gate)
  - » modified MVS model by Dimitri Antoniadis (MIT)
    - – no gate loss
  - » **homotopy needed** to capture -ve res. region

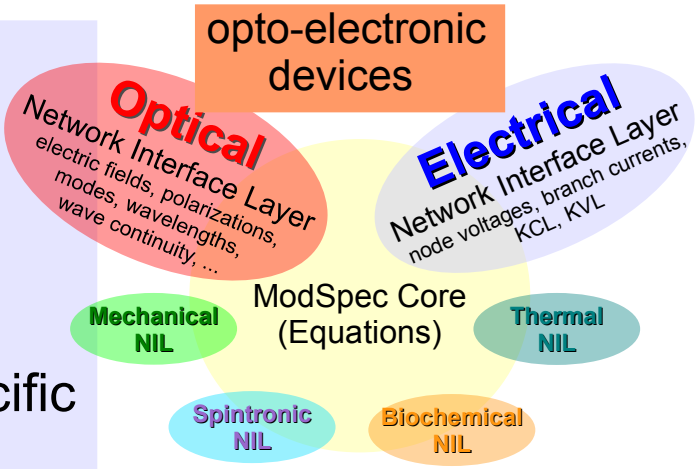




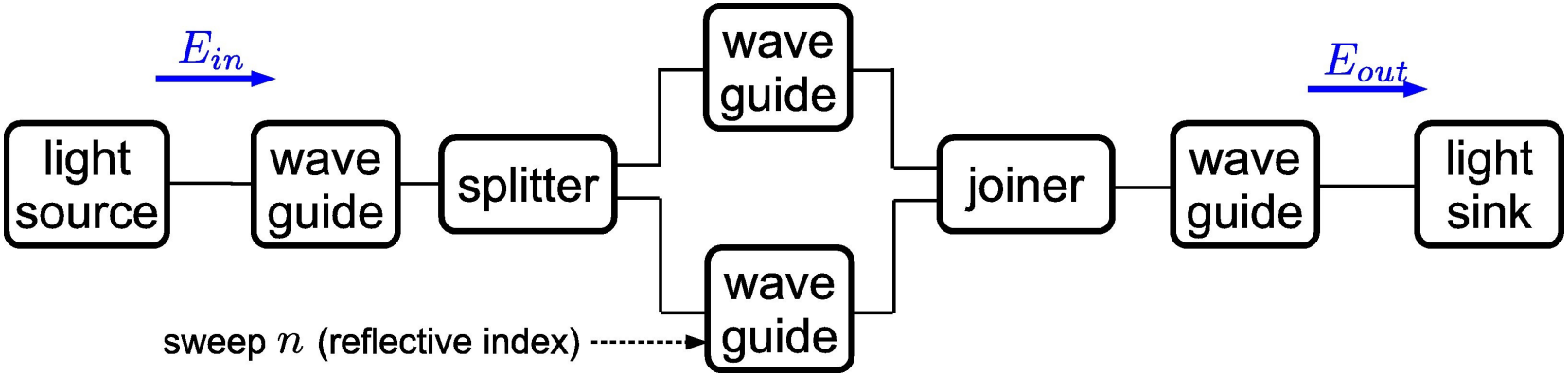
MAPP's Multi-Physics Capabilities

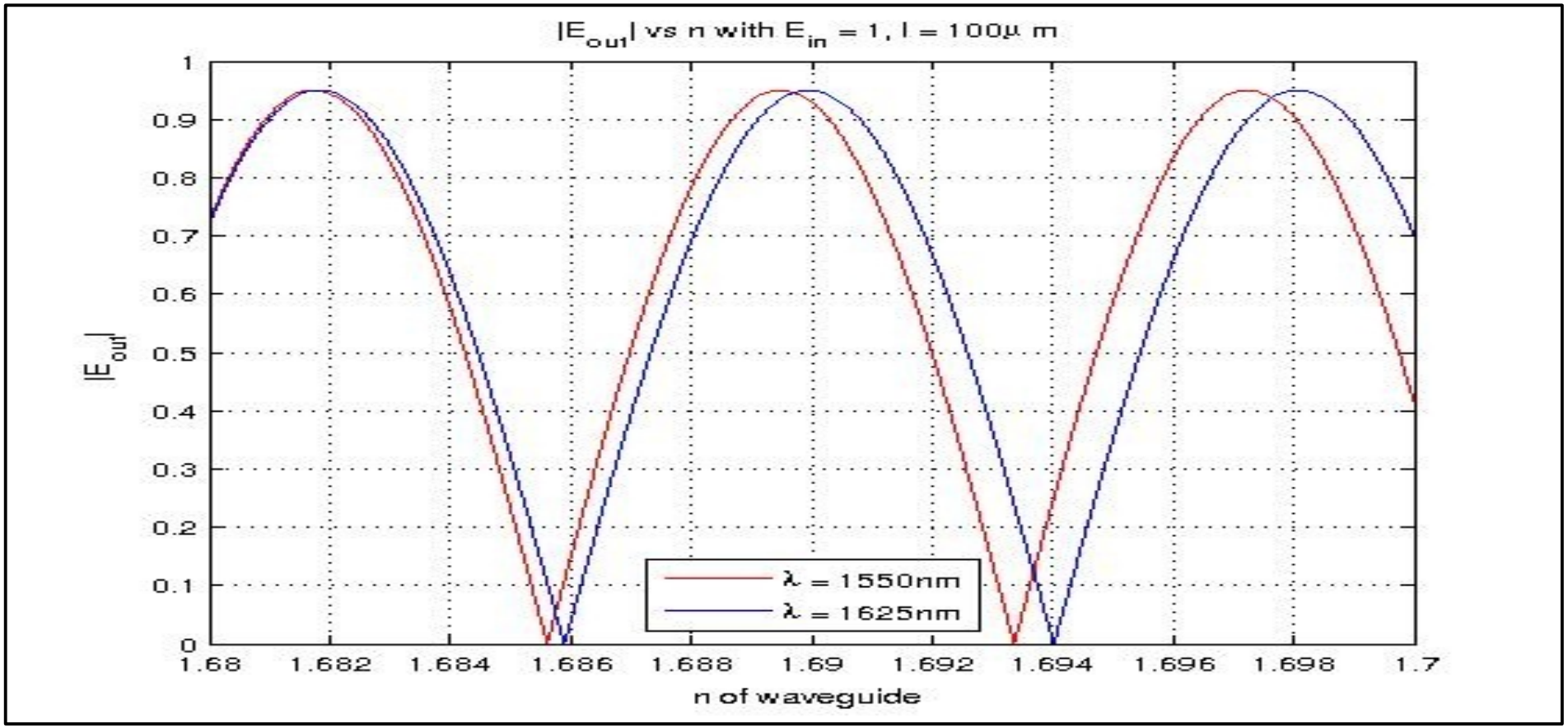


- Networks in different physical domains supported
  - » via **domain-specific Equation Engines**
- Domains interact via devices with multiple **Network Interface Layers (NILs)**
- **Master Equation Engine** orchestrates domain-specific Engines to produce multi-physics system DAEs

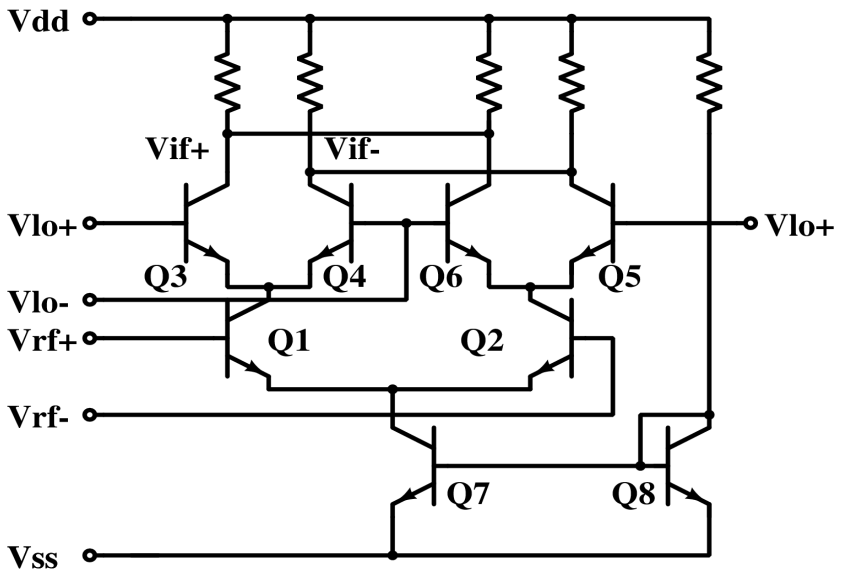


Optical System Modelling/Simulation Example

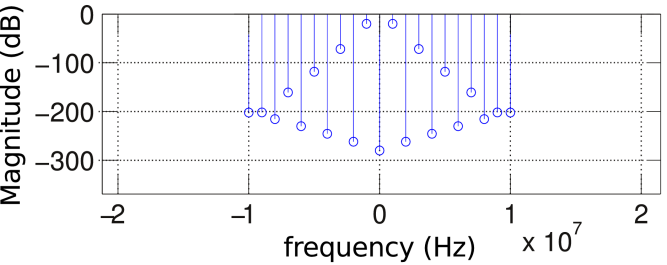


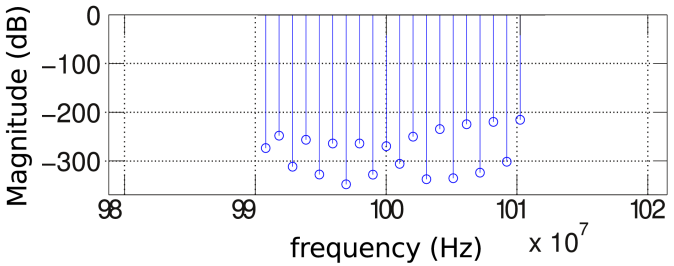


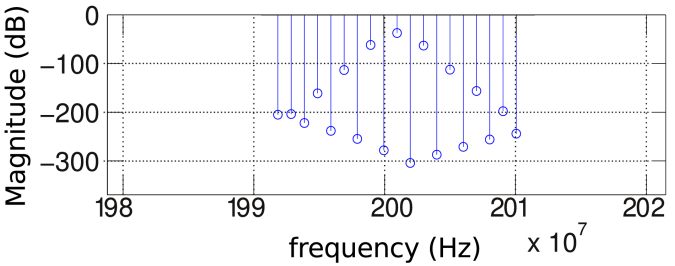
Example: 2-tone HB on Gilbert Cell

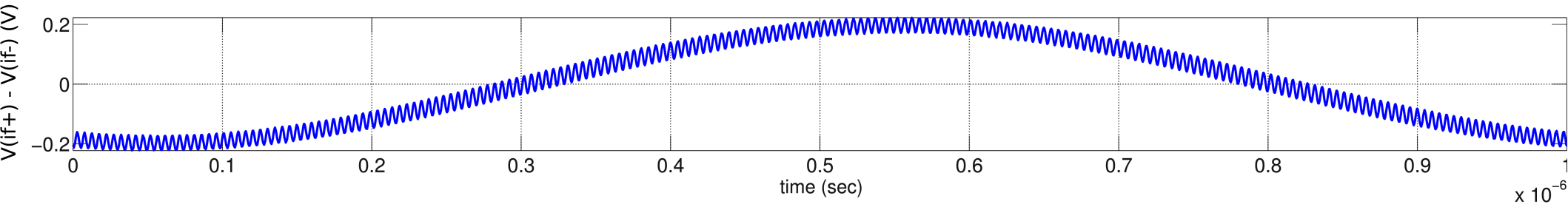


HB on MNA DAE for Gilbert cell

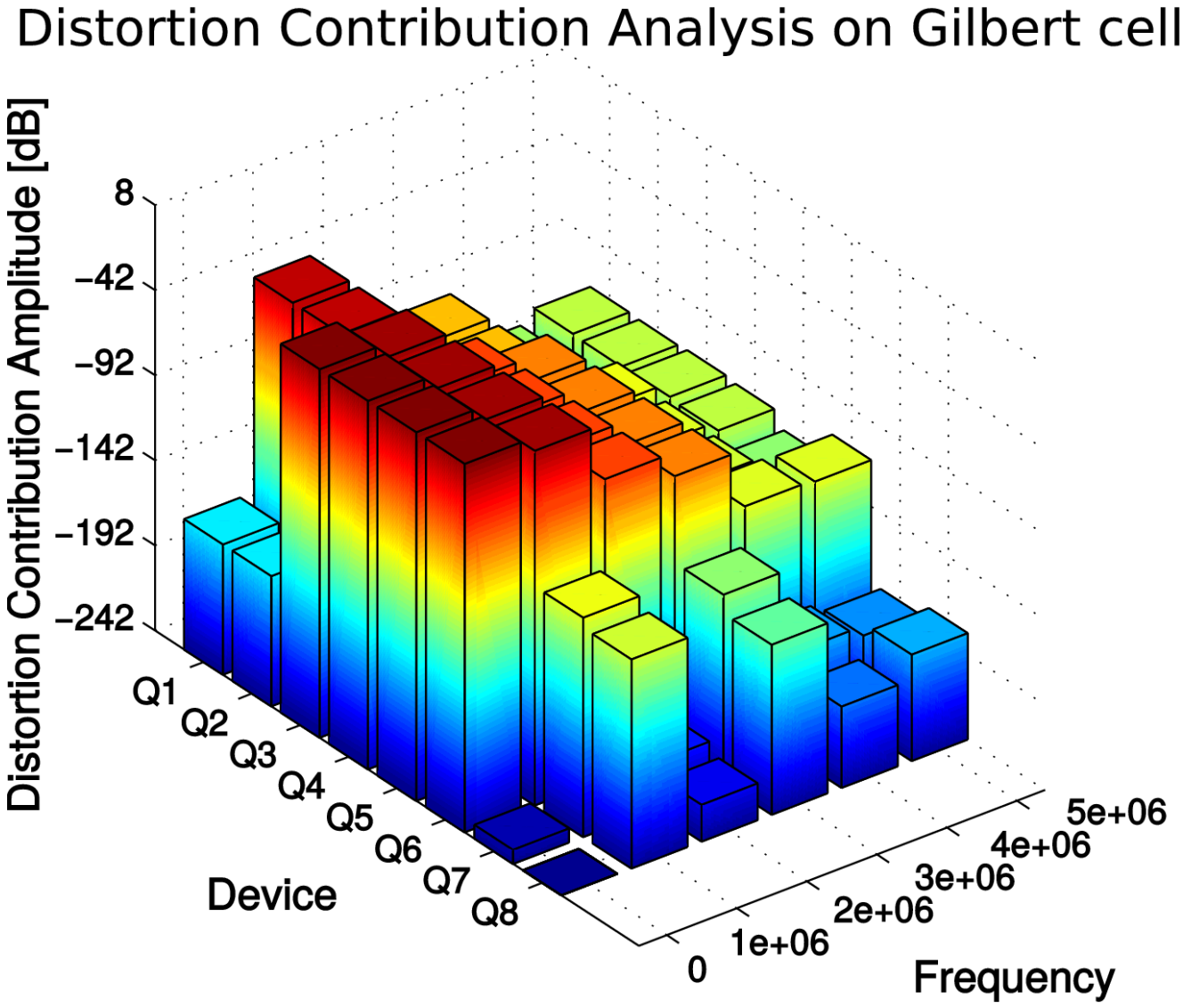




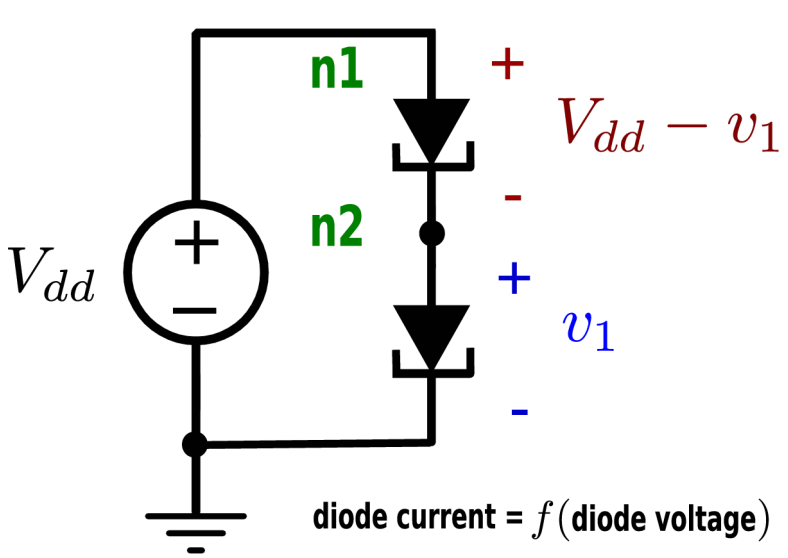


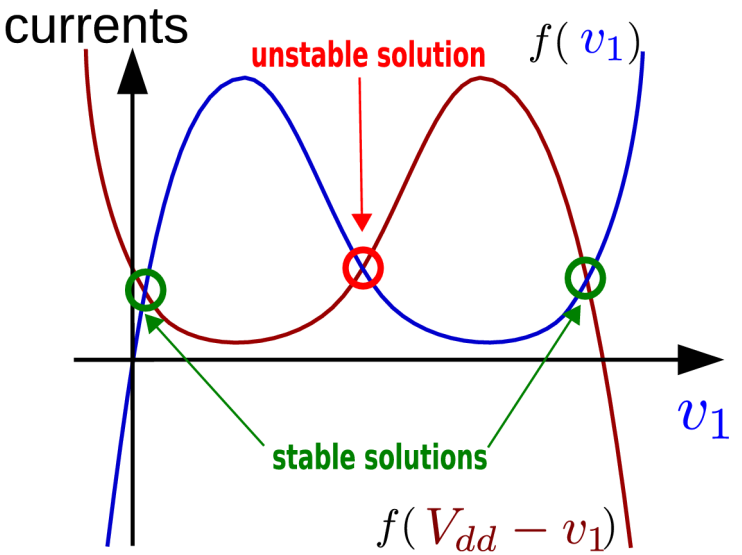


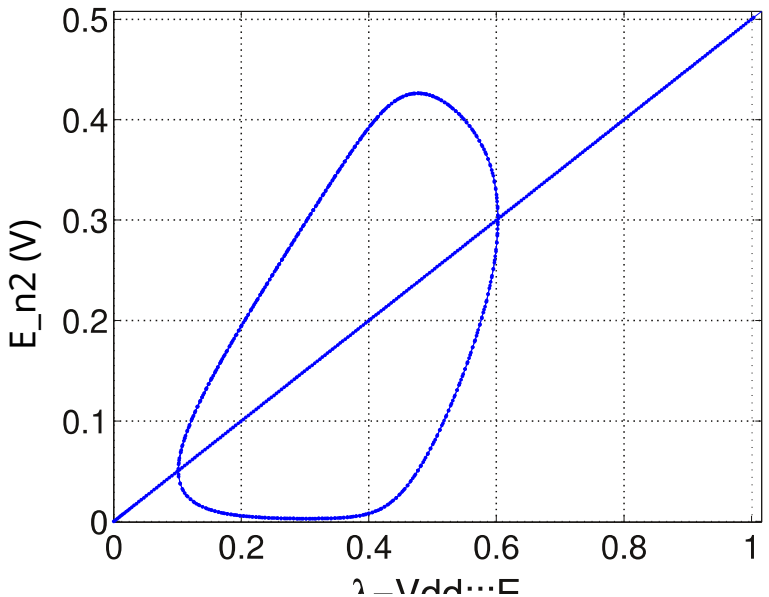
Per-Element Distortion Contribution Analysis

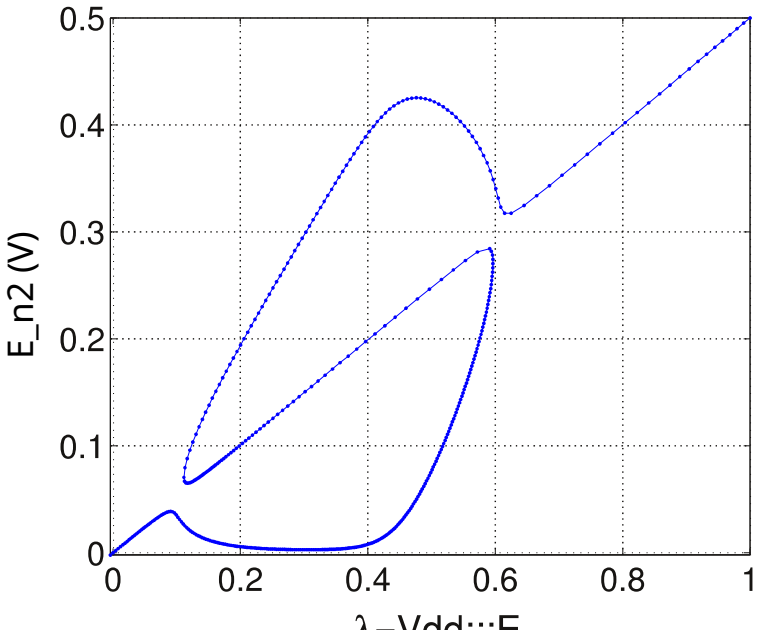


Example: Goto Pair Continuation



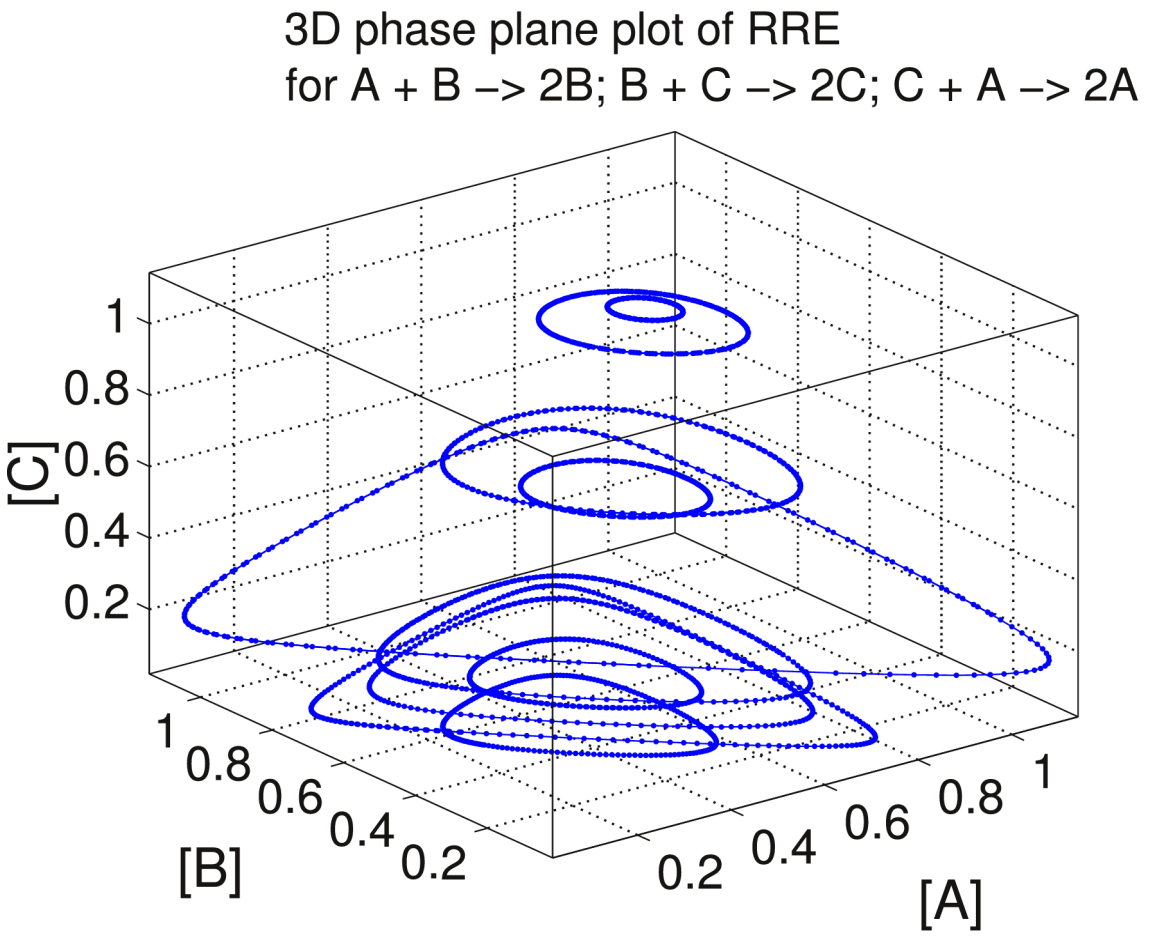




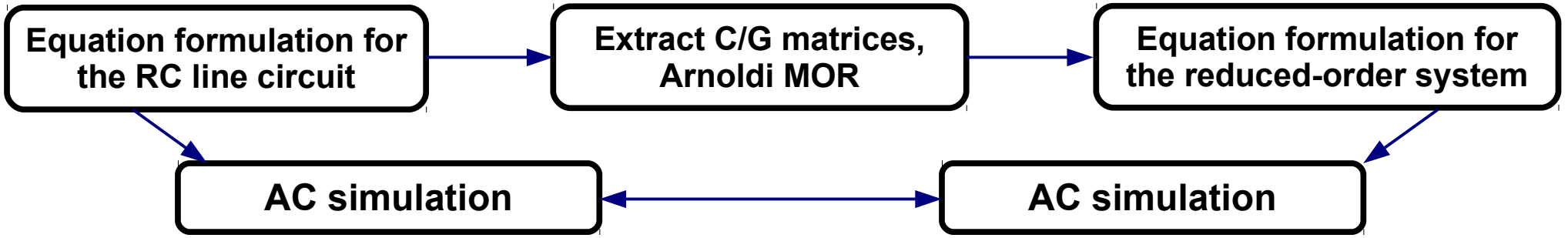


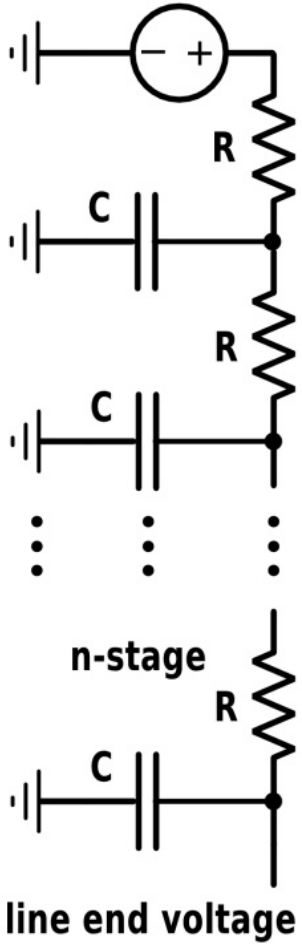
Example: Synthetic Bio Oscillator

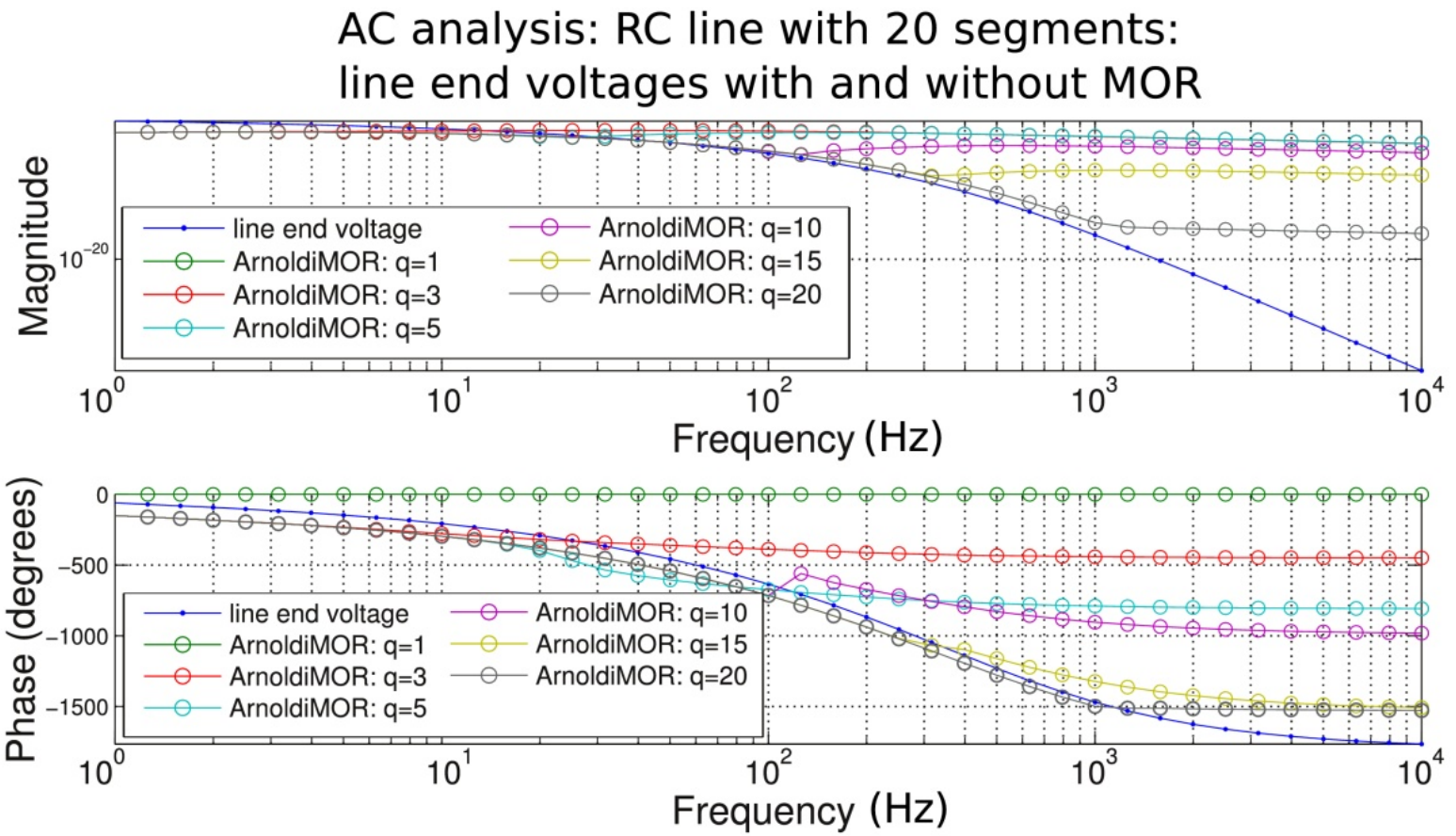
$$A + B \rightharpoonup 2B$$
$$B + C \rightharpoonup 2C$$
$$C + A \rightharpoonup 2A$$



LTI MOR Example in MAPP







Phase-based Reduced-order Model in MAPP

Model osc. in MAPP

not just electrical ones

Simulate PSS: shooting or HB

Extract PPVs

Gen-Adler Analysis

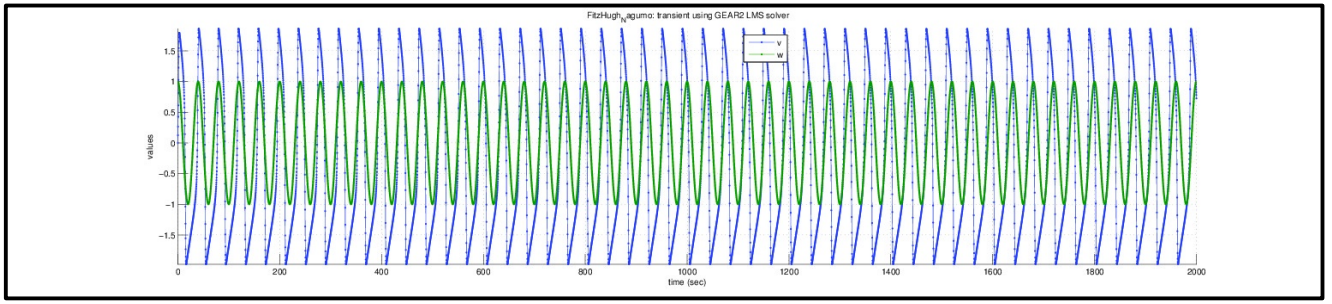
$$\frac{d}{dt}\Delta\phi(t) = f_0 - f_1 + f_0 \cdot g(\Delta\phi(t))$$

DC — locking range

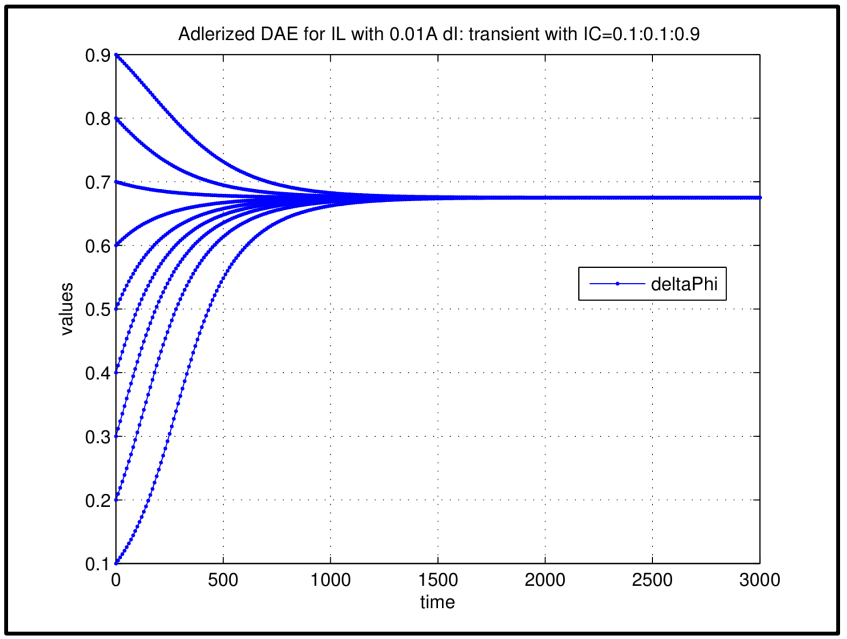
TRAN — phase dynamics

Observe Injection Locking properties

Standard TRAN simulation:



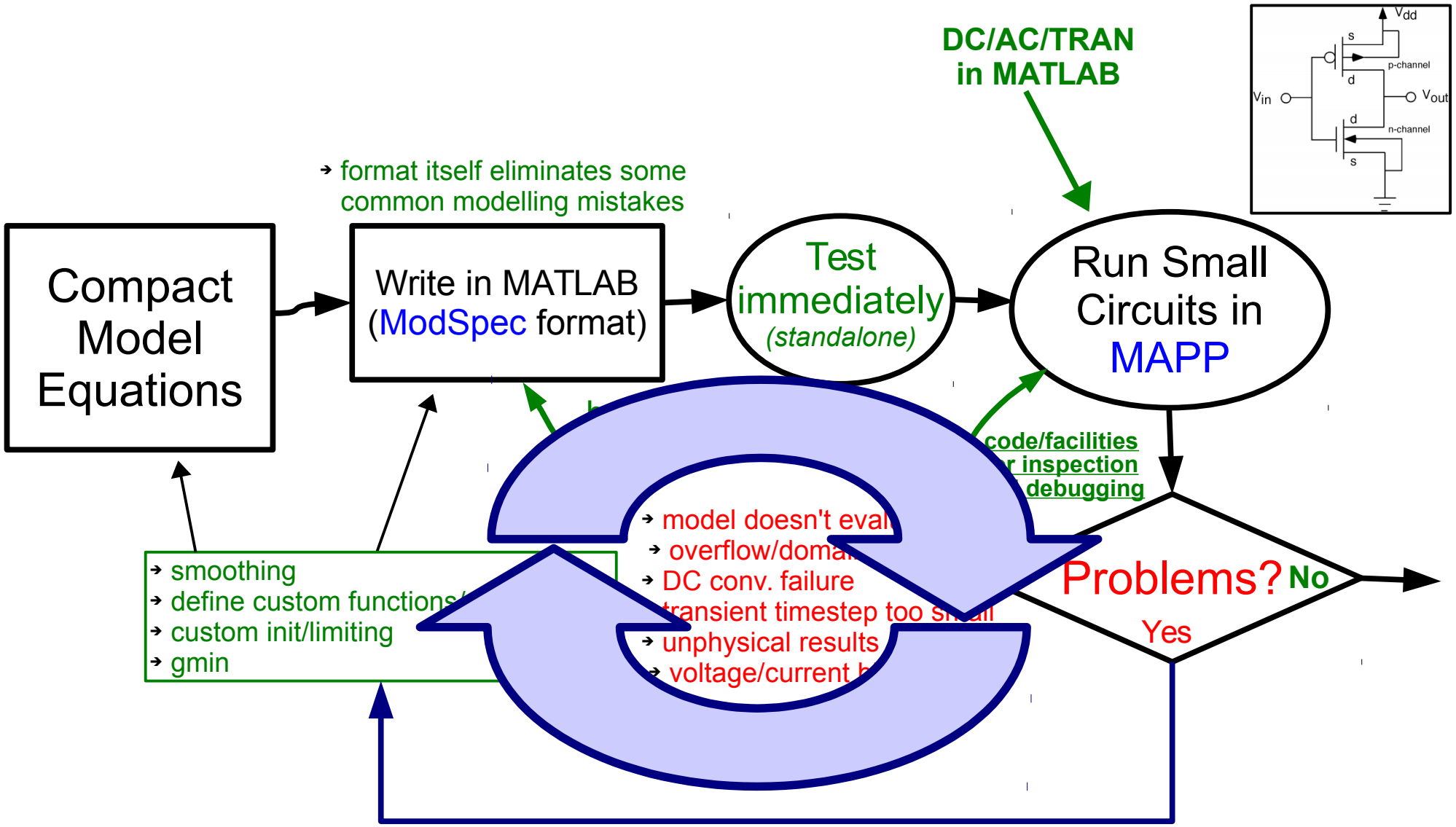
Phase-based TRAN:



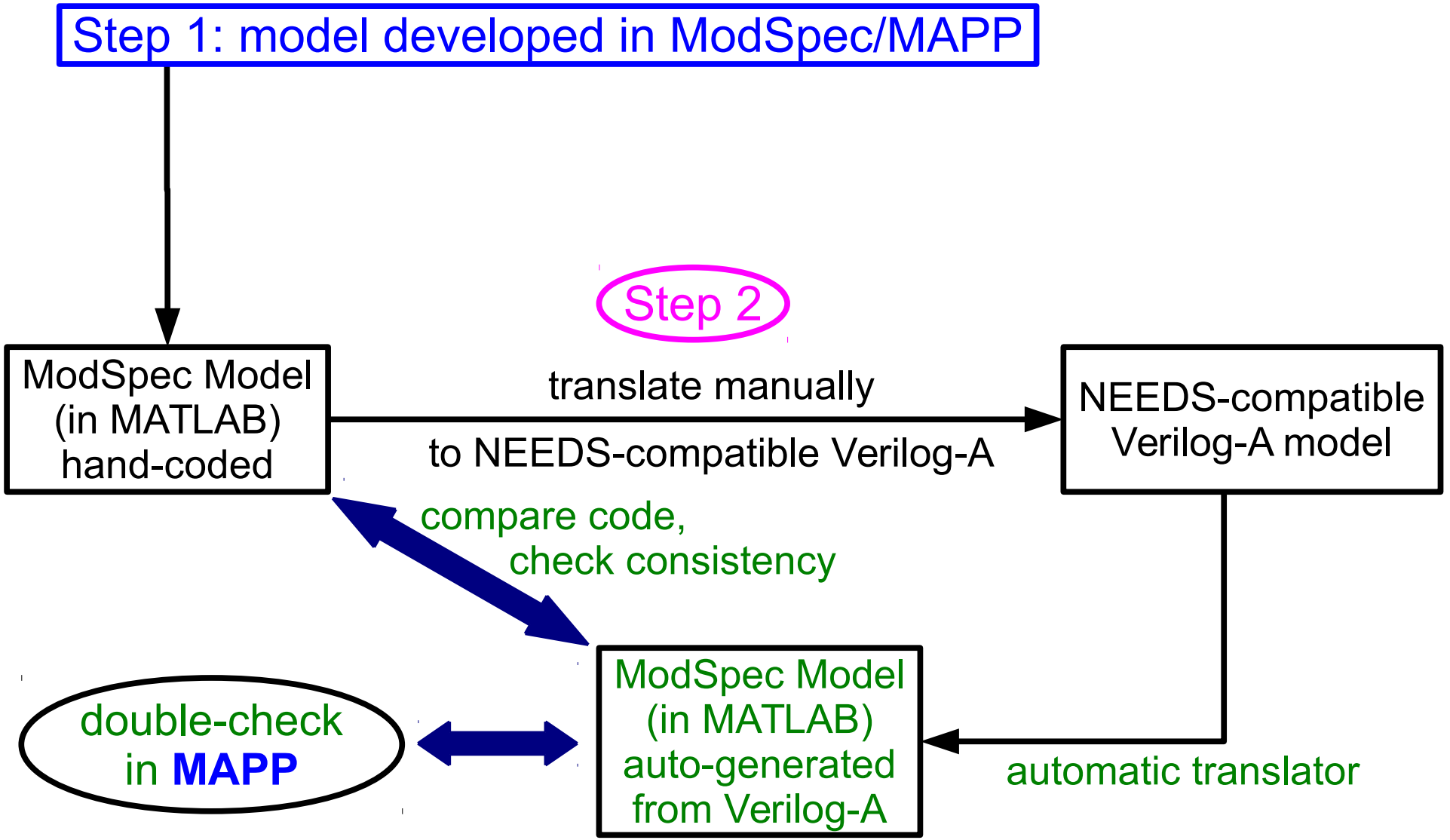
$\Delta\phi(t)$ captures phase response nicely

details: Bhansali/Roychowdhury, "Gen-Adler: the Generalized Adler's equation for injection locking analysis in oscillators". Proc. ASPDAC, 2009.

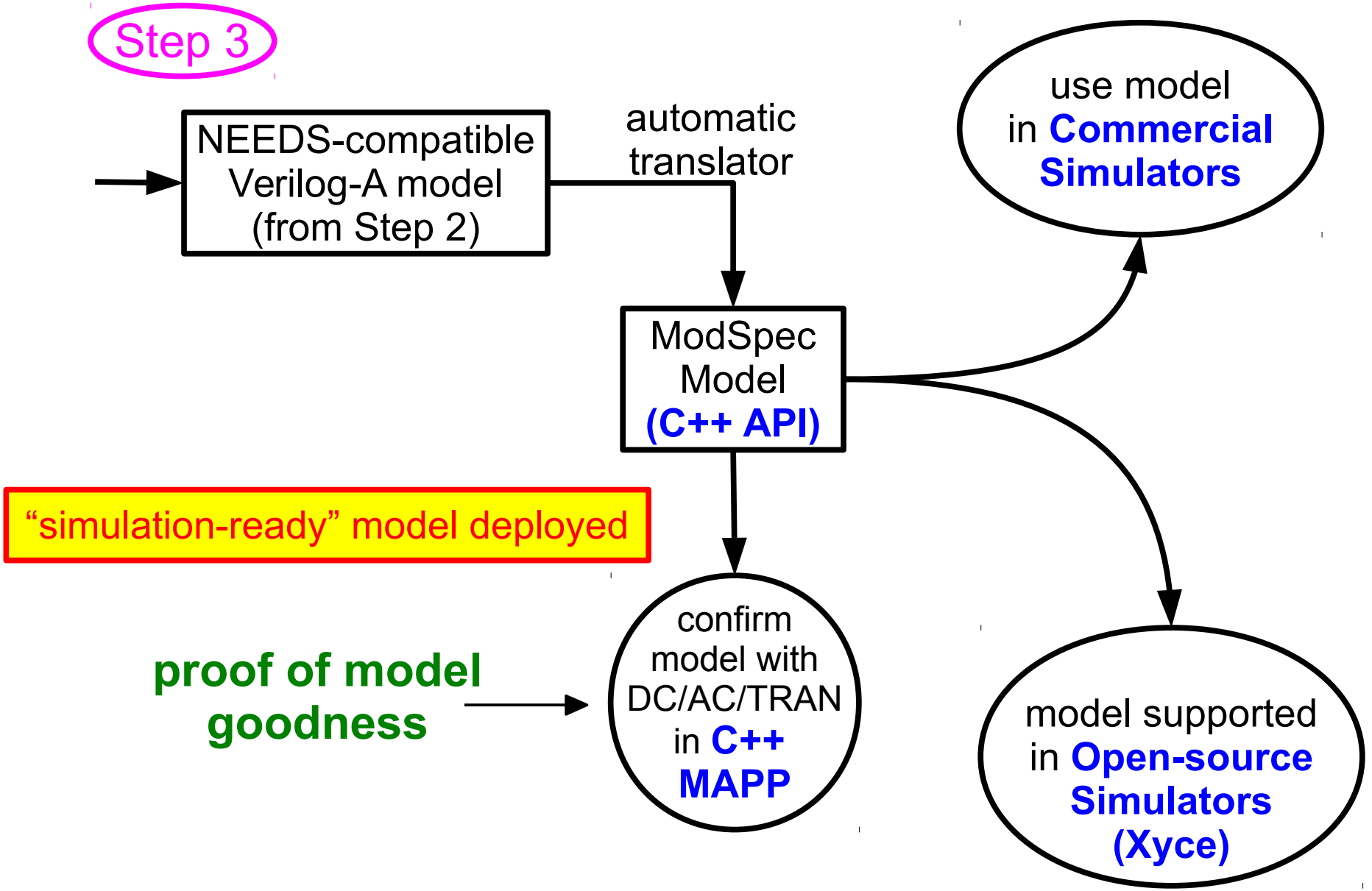
MAPP for Device Model Development



MAPP Model Development Flow (2)



MAPP Model Development Flow (3)



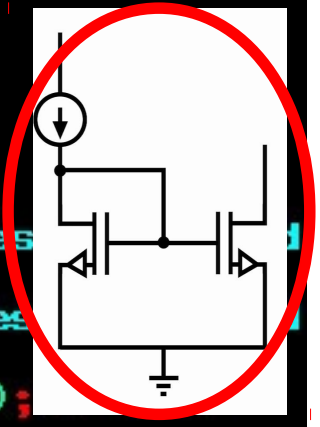
Glimpse: Circuit Netlist in MAPP

```
function cktnetlist = current_mirror_ckt()
    % MDE netlist for a simple current mirror
    % Author: Bichen Wu, 2013/10/22
    cktnetlist.cktname = 'current_mirror';
    cktnetlist.nodenames = {'D1', 'D2', 'M2D', 'G'}; % nodes in the circuit
    cktnetlist.groundnodename = 'gnd';

    Vd = 2; k = 1e-4; IO = 10e-6; RL = 1e4; W = 270; L = 180; DSgmin = 1e-8;

    % define a time-varying function for the voltage source
    utfunc = @(t,args) args.offset + args.A * sin(2*pi*args.f*t);
    utargs.offset = IO; utargs.A = 1e-6; utargs.f = 2e7;

    % set up the netlist
    cktnetlist = add_element(cktnetlist, vsrcModSpec(), 'VD1', ...
        {'D1', 'gnd'}, {}, {{'E', {'DC', Vd}}}); % vsrc between nodes
    cktnetlist = add_element(cktnetlist, vsrcModSpec(), 'VD2', ...
        {'D2', 'gnd'}, {}, {{'E', {'DC', Vd}}}); % vsrc between nodes
    cktnetlist = add_element(cktnetlist, isrcModSpec(), 'IO', ...
        {'D1', 'G'}, {}, {{'I', {'DC', IO},{'TRAN',utfunc,utargs}}});
    cktnetlist = add_element(cktnetlist, resModSpec(), 'R', ...
        {'D2', 'M2D'}, {{'R', RL}}, {}); % resistor between D2 and M2D
    cktnetlist = add_element(cktnetlist, SH_MOS_ModSpec(), 'M1', ... % n-MOS
        {'G', 'G', 'gnd'}, {{'Type', 'N'}, {'Beta', k*W/L}, {'VT', 0.3}}, {});
    cktnetlist = add_element(cktnetlist, SH_MOS_ModSpec(), 'M2', ... % n-MOS
        {'M2D', 'G', 'gnd'}, {{'Type', 'N'}, {'Beta', k*W/L}, {'VT', 0.3}}, {});
end % function current_mirror_ckt
"current_mirror_ckt.m" 27L, 1455C written                    1,1           All
```



Glimpse: Verilog-A → ModSpec Translator

```
// Series RLC
// Version 1a, 1 June 04
// Ken Kundert
//
// Downloaded from The Designer's Guide Community
// (www.designers-guide.org).
// Taken from "The Designer's Guide to Verilog-AMS"
// by Kundert & Zinke. Chapter 3, Listing 14.

`include "disciplines.vams"

module series_rlc2 (p, n);
    parameter real r=1000;      // resistance (Ohms)
    parameter real l=1e-9;      // inductance (H)
    parameter real c=1e-6;      // capacitance (F)
    inout p, n;
    electrical p, n, i;
    branch (p, i) rl, (i, n) cap;

    analog begin
    V(rl) <+ r*I(rl);
    V(rl) <+ ddt(l*I(rl));
    I(cap) <+ ddt(c*V(cap));
    end
endmodule
```

Verilog-A

```
function MOD = series_rlc2_ModSpec_wrapper ()
    MOD = ee_model();
    MOD = add_to_ee_model (MOD, 'terminals', {'p', 'n'});
    MOD = add_to_ee_model (MOD, 'explicit_outs', {'ipn'});
    MOD = add_to_ee_model (MOD, 'internal_unks', {'Irl'});

    MOD = add_to_ee_model (MOD, 'parms', {'r', 1000});
        % resistance (Ohms)
    MOD = add_to_ee_model (MOD, 'parms', {'l', 1e-9});
        % inductance (H)
    MOD = add_to_ee_model (MOD, 'parms', {'c', 1e-6});
        % capacitance (F)
    MOD = add_to_ee_model (MOD, 'fqei_all', @fqei);
    MOD = finish_ee_model(MOD);
end

function [Vrl, Frl, Icap, Qcap] = series_rlc2(Irl, Vcap, r, l, c)
    Vrl = r*Irl;
    Frl = l*Irl;
    Icap = 0;
    Qcap = c*Vcap;
end

function [fe, qe, fi, qi] = fqei(S)
    v2struct(S);

    Vcap = vpn;
    [Vrl, Frl, Icap, Qcap] = series_rlc2(Irl, Vcap, r, l, c);

    fe(1,1) = Irl + Icap;
    qe(1,1) = 0 + Qcap;
    fi(1,1) = vpn - Vrl;
    qi(1,1) = 0 - Frl;
end
```

ModSpec (MATLAB)

Glimpse: ModSpec Model in Xyce

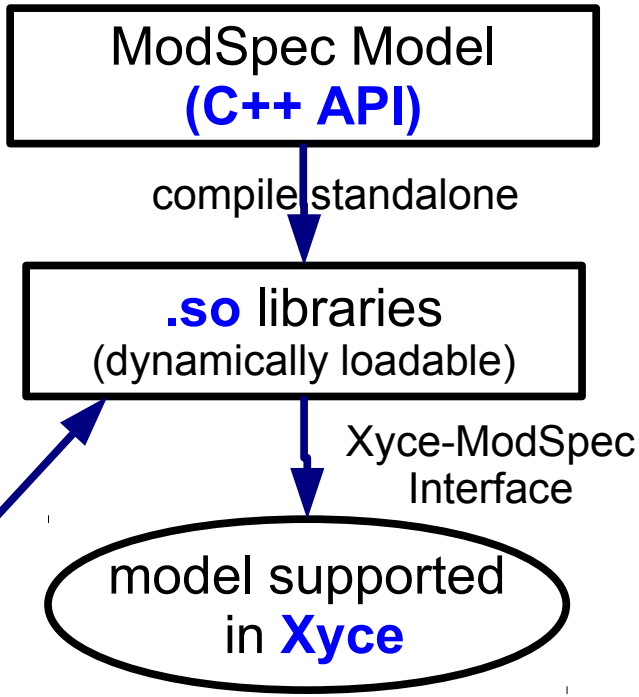
```
 1 *** Test-bench for generting dc response of an inverter
 2
 3 *** Creat sub-circuit for the inverter
 4 .subckt inverter Vin Vout Vvdd Vgnd
 5
 6 yModSpec_Device X1 Vvdd Vin Vout Vvdd MVSmod type=-1 W=1.0e-4
 7    Lgdr=32e-7 dLg=8e-7 Cg=2.57e-6 beta=1.8 alpha=3.5 Tjun=300
 8    Cif = 1.38e-12 Cof=1.47e-12 phib=1.2 gamma=0.1 mc=0.2
 9    CTM_select=1  Rs0=100 Rd0 = 100 n0=1.68 nd=0.1 vxo=7542204
10    mu=165 Vt0=0.5535 delta=0.15
11
12 yModSpec_Device X0 Vout Vin Vgnd Vgnd MVSmod type=1 W=1e-4
13    Lgdr=32e-7 dLg=9e-7 Cg=2.57e-6 beta=1.8 alpha=3.5 Tjun=300
14    Cif=1.38e-12 Cof=1.47e-12 phib=1.2 gamma=0.1 mc=0.2
15    CTM_select=1 Rs0=100 Rd0=100 n0=1.68 nd=0.1 vxo=1.2e7
16    mu=200 Vt0=0.4 delta=0.15
17
18 .model MVSmod MODSPEC_DEVICE SONAME=MVS_ModSpec_Element.so
19
20 .ends
21
22 *** circuit layout
23 Vsup sup 0 1
24 Vin in 0 0
25 Vsource source 0 0
26 X2  in out sup 0 inverter
27
28 *** simulation
29 .dc Vin 0 1 0.01
30
31 .print dc V(in)
32 *** END
33 .end
```

.model line

model's name

model parameter:
name of .so library

Xyce netlist for inverter
(using MVS ModSpec/C++ model)



DC sweep using Xyce

Glimpse: Running Analyses in MAPP

1. convert cktnetlist to DAE:

```
DAE = MNA_EqnEngine(cktnetlist);
```

2. run DC operating point analysis:

```
dcop = dot_op(DAE);
dcop.print(dcop);
```

3. run transient simulation:

```
xinit = zeros(DAE.nunks(DAE), 1); xinit(2) = 0.3;
tstart = 0; tstep = 0.1e-9; tstop = 50e-9;
tranObj = dot_transient(DAE, xinit, tstart, tstep, tstop);
tranObj.plot(tranObj);
```

MAPP: Public Release

- Open Source download: http://mapp.eecs.berkeley.edu
  - » mailing list (MAPP announcements/discussion)
  - » bug reporting and tracking site
  - » git repository access (you can contribute)
- License
  - » primary: GPL-v3
  - » alternative licensing available
    - – eg, SRC contract terms apply for SRC company use
  - » contributors can specify their own alternative licensing terms for their contributions

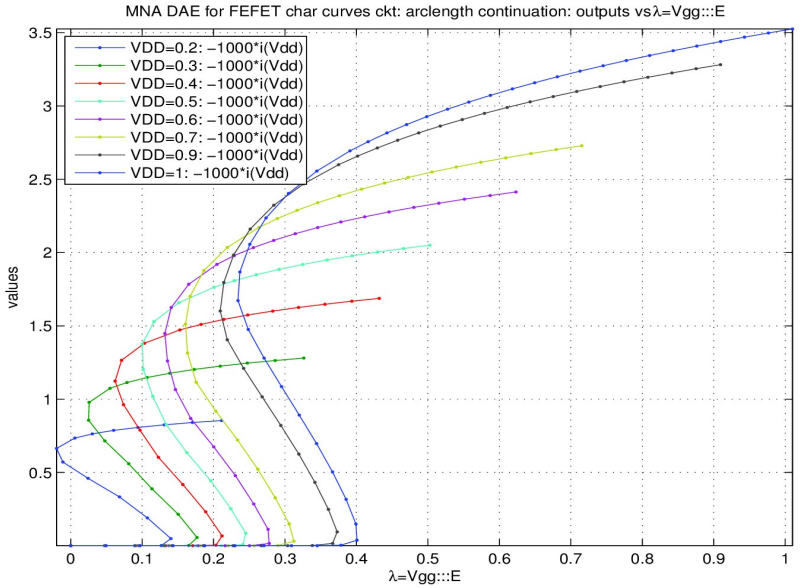
MAPP: Features

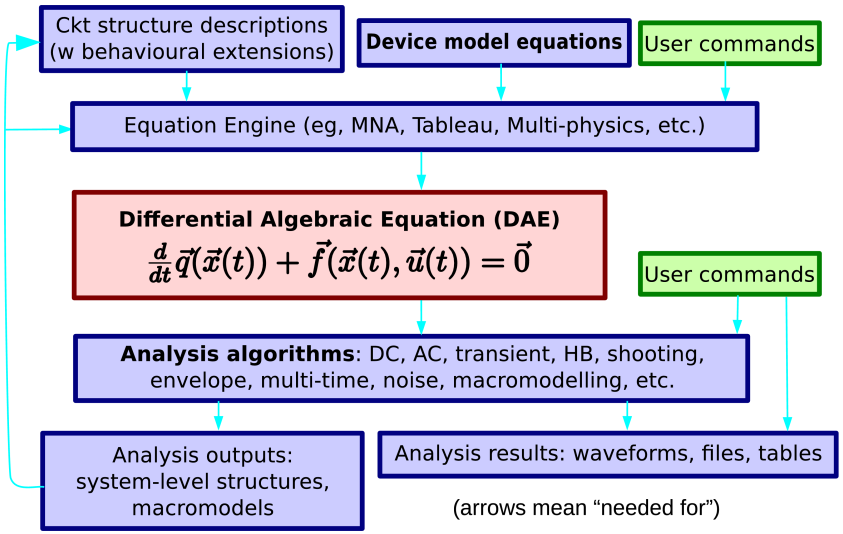
- Works entirely in MATLAB
  - » C++ interfaces in the works
- Help system (start with `help MAPP`)
  - » quick start walk-through
- Automatic differentiation (vecvalder)
  - » `help MAPPautodiff`
- Executable device specification (ModSpec)
  - » examples, tutorial: part of help
- DC, AC, transient analyses
  - » also noise, homotopy, HB, shooting, PPV, MOR, etc. (not released yet)
- Automated testing system exercising suite of tests

MAPP: Intended Uses

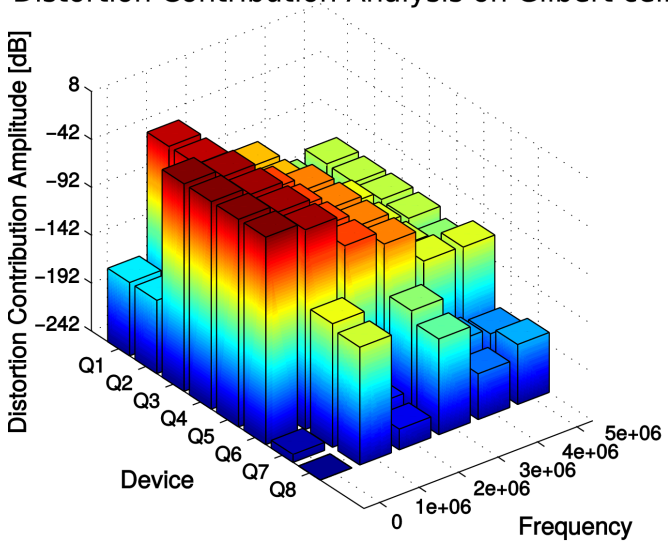
- Developing simulation-ready device models
  - » including multi-physics devices, network connectivity
- Quickly prototyping new simulation algorithms
  - » hours/days to implement a new analysis
    - – assess strengths/limitations before investing resources to implement in “real simulators”
- Learning or teaching modelling/simulation
  - » MATLAB → broadly accessible
  - » help system, tutorials, supporting resources

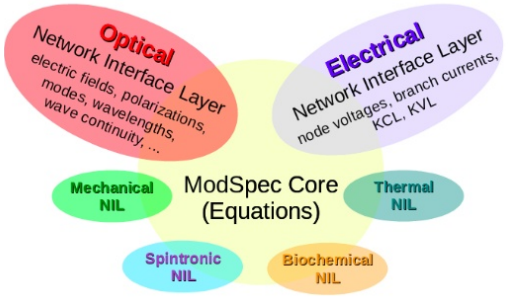
Summary



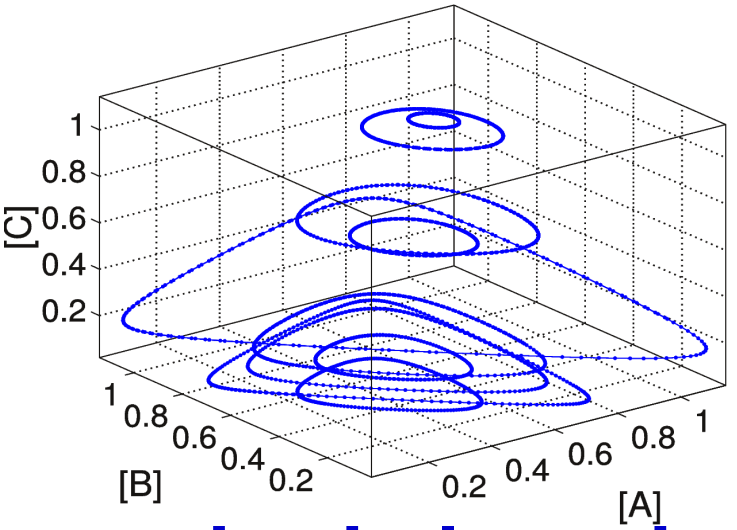


Distortion Contribution Analysis on Gilbert cell





3D phase plane plot of RRE
for A + B –> 2B; B + C –> 2C; C + A –> 2A



http://MAPP.eecs.berkeley.edu