MAPP:
The Berkeley Model and Algorithm Prototyping Platform

Tianshi Wang, Karthik Aadithya, A. Gokcen Mahmutoglu
Bichen Wu, Jian Yao and Jaijeet Roychowdhury

EECS Department
University of California at Berkeley
Model Development Today

- Device physics into mathematical code (MATLAB)
- Interdependent with simulation
  - Check for errors
  - Does it converge?
  - Can it replicate real dynamical device behavior?
- Translate into Verilog-A → Run with HSpice or Spectre
- What if there are problems?
  - Simulators: Complicated under the hood
  - Makes testing and debugging hard
- MAPP: specifically designed with model development in mind
  - All purpose model development in pure MATLAB
Verilog-A-based Model Development Flow

- Device physics
- Analytical Equations for Device
Verilog-A-based Model Development Flow

Device physics

Analytical Equations for Device

Write Verilog-A Model
Verilog-A-based Model Development Flow

1. Device physics
   Analytical Equations for Device

2. Write Verilog-A Model

3. Run Circuits in **Commercial Simulators**
Verilog-A-based Model Development Flow

Device physics

Analytical Equations for Device

Write Verilog-A Model

Run Circuits in Commercial Simulators

robust convergence?
Verilog-A-based Model Development Flow

- Device physics
- Analytical Equations for Device
- Write Verilog-A Model
- Run Circuits in Commercial Simulators

robust convergence?

Yes

Deploy model
Verilog-A-based Model Development Flow

1. **Device physics**
   - Analytical Equations for Device

2. **Write Verilog-A Model**

3. **Run Circuits in Commercial Simulators**

4. **Deploy model**

   **convergence vs physics tradeoff**

   - robust convergence?
     - Yes

---

T. Wang and J. Roychowdhury, University of California at Berkeley
Verilog-A-based Model Development Flow

Device physics

Analytical Equations for Device

Write Verilog-A Model

Run Circuits in Commercial Simulators

support for convergence hooks limited/missing

convergence vs physics tradeoff

robust convergence?

Deploy model

Yes
Verilog-A-based Model Development Flow

1. Device physics
2. Analytical Equations for Device
3. Write Verilog-A Model
4. Run Circuits in Commercial Simulators
   - Limited/missing support for convergence hooks
   - No ability to debug code
   - Convergence vs physics tradeoff
5. Deploy model

Robust convergence?
Three Steps of MAPP

- **First Step**
  
  Create correct, robust model in pure MATLAB

- **Second Step**
  
  Create and verify Verilog-A model: VAPP

- **Third Step**
  
  Convert into fast C++ code: MAPP C++ API, Xyce-ModSpec interface
Compact Model Equations
MAPP for Device Model Development

- format itself eliminates some common modelling mistakes

```
Compact Model Equations

Write in MATLAB (ModSpec format)
```
MAPP for Device Model Development

Compact Model Equations → Write in MATLAB (ModSpec format) → Test immediately (standalone)

- format itself eliminates some common modelling mistakes

basic debug
MAPP for Device Model Development

Compact Model Equations → Write in MATLAB (ModSpec format) → Test immediately (standalone) → Run Small Circuits in MAPP

- format itself eliminates some common modelling mistakes

DC/AC/TRAN in MATLAB

basic debug
MAPP for Device Model Development

- Compact Model Equations
- Write in MATLAB (ModSpec format)
- Test immediately (standalone)
- Run Small Circuits in MAPP
- DC/AC/TRAN in MATLAB

format itself eliminates some common modelling mistakes

basic debug

Problems?
MAPP for Device Model Development

Compact Model Equations

Write in MATLAB (ModSpec format)

Test immediately (standalone)

Run Small Circuits in MAPP

DC/AC/TRAN in MATLAB

format itself eliminates some common modelling mistakes

basic debug

model doesn't evaluate
overflow/domain errors
DC conv. failure
transient timestep too small
unphysical results
voltage/current blows up

Problems?
Yes

code/facilities for inspection and debugging
MAPP for Device Model Development

- Compact Model Equations
  - Write in MATLAB (ModSpec format)
    - Test immediately (standalone)
      - Run Small Circuits in MAPP
        - Problems?
          - Yes
            - code/facilities for inspection and debugging
              - basic debug
                - model doesn't evaluate
                - overflow/domain errors
                - DC conv. failure
                - transient timestep too small
                - unphysical results
                - voltage/current blows up
              - format itself eliminates some common modelling mistakes

A. Gokcen Mahmutoglu, amahmutoglu@berkeley.edu
MAPP for Device Model Development

Compact Model Equations

Write in MATLAB (ModSpec format)

Test immediately (standalone)

Run Small Circuits in MAPP

Problems?

Yes

- format itself eliminates some common modelling mistakes
- model doesn't converge
- overflow/domain errors
- DC conv. failure
- transient timestep too small
- unphysical results
- voltage/current blows up
- smoothing
- define custom functions/derivatives
- custom init/limiting
- gmin

DC/AC/TRAN in MATLAB

code/facilities for inspection and debugging
MAPP for Device Model Development

Compact Model Equations

Write in MATLAB (ModSpec format)

Test immediately (standalone)

Run Small Circuits in MAPP

Problems?

No

Yes

- format itself eliminates some common modelling mistakes
- smoothing
- define custom functions/derivatives
- custom init/limiting
- gmin

DC/AC/TRAN in MATLAB

- model doesn't compile
- overflow/domain errors
- DC conv. failure
- transient timestep too small
- unphysical results
- voltage/current blows up

- code/facilities for inspection and debugging

A. Gokcen Mahmutoglu, amahmutoglu@berkeley.edu
Glimpse: Diode Model in MAPP

```matlab
function MOD = diodeCapacitor_ModSpec_wrapper ()
% This function returns a ModSpec model for a simple diode with capacitor
 MOD = ee_model();
 MOD = add_to_ee_model(MOD, 'external_nodes', {p, n});
 MOD = add_to_ee_model(MOD, 'explicit_outs', {ipn});
 MOD = add_to_ee_model(MOD, 'parms', {'C', 2e-12, 'Is', 1e-12, 'VT', 0.025});
 MOD = add_to_ee_model(MOD, 'f', @f);
 MOD = add_to_ee_model(MOD, 'q', @q);
 MOD = finish_ee_model(MOD);
 end

function out = f(S)
 %vzstruct(S);
 out = Is*(exp(vpn/VT)-1);
 end

function out = q(S)
 %vzstruct(S);
 out = C*vpn;
 end
```
Glimpse: Diode Model in MAPP

```matlab
function MOD = diodeCapacitor_ModSpec_wrapper ()
    MOD = add_to_ee_model(MOD, 'external_nodes', {'p', 'n'});
    MOD = add_to_ee_model(MOD, 'explicit_outs', {'ipn'});
    MOD = add_to_ee_model(MOD, 'parms', {'C', 2e-12, 'Is', 1e-12, 'VT', 0.025});
    MOD = add_to_ee_model(MOD, 'f', @f);
    MOD = add_to_ee_model(MOD, 'q', @q);
    MOD = finish_ee_model(MOD);
end

function out = f(S)
    v2struct(S);
    out = Is*(exp(vpn/VT)-1);
end

function out = q(S)
    v2struct(S);
    out = C*vpn;
end
```
function MOD = diodeCapacitor_ModSpec_wrapper()
% This function returns a ModSpec model for a simple diode with capacitor
MOD = ee_model();
MOD = add_to_ee_model(MOD, 'external_nodes', {'p', 'n'});
MOD = add_to_ee_model(MOD, 'explicit_outs', {'ipn'});
MOD = add_to_ee_model(MOD, 'pars', {'C', 2e-12, 'Is', 1e-12, 'VT', 0.025});
MOD = add_to_ee_model(MOD, 'f', @f);
MOD = add_to_ee_model(MOD, 'q', @q);
MOD = finish_ee_model(MOD);
function out = f(S)
out = Is*(exp(vpn/VT)-1);
end
function out = q(S)
out = C*vpn;
end
Glimpse: Diode Model in MAPP

```matlab
function MOD = diodeCapacitor_ModSpec_wrapper ()
%function MOD = diodeCapacitor_ModSpec_wrapper()
% This function returns a ModSpec model for a simple diode with capacitor
MOD = ee_model();

MOD = add_to_ee_model(MOD, 'external_nodes', {'p', 'n'});
MOD = add_to_ee_model(MOD, 'explicit_outs', {'ipn'});
MOD = add_to_ee_model(MOD, 'params', {'C', 2e-12, 'Is', 1e-12, 'VT', 0.025});
MOD = add_to_ee_model(MOD, 'f', @f);
MOD = add_to_ee_model(MOD, 'q', @q);
MOD = finish_ee_model(MOD);
end

function out = f(S)
out = struct(S);
out = Is*(exp(vpn/VT)-1);
end

function out = q(S)
out = struct(S);
out = C*vpn;
end
```
Glimpse: Diode Model in MAPP

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

...
Glimpse: Diode Model in MAPP

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

A. Gokcen Mahmutoglu, amahmutoglu@berkeley.edu
MAPP: Compact Model Prototyping

(a) BSIM6.1.0
default: $L=10\mu m$, $W=10\mu m$

(b) PSP Level 103 v3.0
default: $L=10\mu m$, $W=10\mu m$

(c) MVS v1.0.1
default: $L=80\text{nm}$, $W=1\mu m$

(d) MOS11 v2
default: $L=1\mu m$, $W=1\mu m$
FEFET: Char Curves, Inverter

- Ferro-electrical FET (negative cap. gate)
  » modified MVS model by Dimitri Antoniadis (MIT)
    - Unpublished work
  » **homotopy needed** to capture -ve res. region

![Graph showing char curves for FEFET](image-url)
FEFET: Char Curves, Inverter

- Ferro-electrical FET (negative cap. gate)
  - modified MVS model by Dimitri Antoniadis (MIT)
    - Unpublished work
  - homotopy needed to capture -ve res. region
Landau-Lifshitz-Gilbert (LLG) Model: Spin-Torque Devices

Three dimensional oscillator – rotation of magnetization in response to torques
MAPP's Multi-Physics Capabilities

- laser
- multi-mode fiber
- splitter
- Mach-Zehnder electro-optical modulator
- combiner
- multi-mode fiber
- photo-detector

Transmitter

Receiver
MAPP's Multi-Physics Capabilities
MAPP's Multi-Physics Capabilities

Electronics  Optics  Optics  Electronics

multi-mode fiber  Mach-Zehnder electro-optical modulator  multi-mode fiber
laser  splitter  combiner

Transmitter

Receiver

A. Gokcen Mahmutoglu, amahmutoglu@berkeley.edu
MAPP's Multi-Physics Capabilities

- Networks in different physical domains supported
  - via **domain-specific Equation Engines**
MAPP's Multi-Physics Capabilities

- Networks in different physical domains supported
  - via **domain-specific Equation Engines**

**Diagram:**
- Laser
- Splitter
- Mach-Zehnder electro-optical modulator
- Combiner
- Photo-detector
- Multi-mode fiber
- Opto-Electronics
- Electronics
- Optics
- Electronics
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)**
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)**
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

![Diagram of multi-physical system](image)
MAPP Model Development Flow (2)

Step 1: model developed in ModSpec/MAPP

ModSpec Model (in MATLAB) hand-coded
MAPP Model Development Flow (2)

Step 1: model developed in ModSpec/MAPP

ModSpec Model (in MATLAB) hand-coded

Step 2

translate manually to NEEDS-compatible Verilog-A

NEEDS-compatible Verilog-A model
MAPP Model Development Flow (2)

**Step 1:** model developed in ModSpec/MAPP

- **ModSpec Model (in MATLAB) hand-coded**

**Step 2**

- translate manually to NEEDS-compatible Verilog-A

- **NEEDS-compatible Verilog-A model**

- **ModSpec Model (in MATLAB) auto-generated from Verilog-A**

- automatic translator
MAPP Model Development Flow (2)

Step 1: model developed in ModSpec/MAPP

ModSpec Model (in MATLAB)
hand-coded

translate manually

do double-check
in MAPP

NEEDS-compatible Verilog-A model

Step 2

code, check consistency

compare code, check consistency

ModSpec Model (in MATLAB)
auto-generated from Verilog-A

automatic translator
MAPP Model Development Flow (2)

**Step 1**: model developed in ModSpec/MAPP

- **ModSpec Model (in MATLAB)**
  - hand-coded

**Step 2**

- translate manually to NEEDS-compatible Verilog-A
- compare code, check consistency
- automatic translator

- **ModSpec Model (in MATLAB)**
  - auto-generated from Verilog-A

**end of Step 2**: high level of confidence Verilog-A model is correct/debugged
MAPP Model Development Flow (2)

Step 1: model developed in ModSpec/MAPP

ModSpec Model (in MATLAB) hand-coded

Step 2

NEEDS-compatible Verilog-A model

translate manually to NEEDS-compatible Verilog-A

double-check in MAPP

compare code, check consistency

ModSpec Model (in MATLAB) auto-generated from Verilog-A

end of Step 2: high level of confidence Verilog-A model is correct/debugged
Verilog-A to ModSpec Translation

- Verilog-A Model
- VAPP
- ModSpec Model (MATLAB)
Verilog-A to ModSpec Translation

- DAE formulation
- Semantic checking
- Check good coding practices
- Auto Diff or hardcoded Jacobians
Verilog-A to ModSpec Translation

Verilog-A → VAPP → ModSpec
Verilog-A to ModSpec Translation

Verilog-A → AST → IR-1 → IR-2 → ModSpec
Verilog-A to ModSpec Translation

\[ I(p,n) <+ G \times V(p,n) \]
Verilog-A to ModSpec Translation

Verilog-A → AST → IR-1 → IR-2 → ModSpec

I(p,n) <+ G*V(p,n)

- Start with tokens
Verilog-A to ModSpec Translation

I(p,n) <= G*V(p,n)

- Start with tokens
- Contract representation into more abstract direction.
- Create objects and link them
Verilog-A to ModSpec Translation

\[ I(p,n) \leftrightarrow G \cdot V(p,n) \]

- Start with tokens
- Contract representation into more abstract direction.
- Create objects and link them
- Top level objects directly related to ModSpec internals

**Model**
- IOs
-Parms
-Terminals
-print

**Contribution**
- lhsIO
-rhsIOs
-IsExplicit
-DiffEqn
-print

A. Gokcen Mahmutoglu, amahmutoglu@berkeley.edu
Verilog-A to ModSpec Translation

I(p,n) <+ G*V(p,n)

- Start with tokens
- Contract representation into more abstract direction.
- Create objects and link them
- Top level objects directly related to ModSpec internals
Glimpse: Verilog-A → ModSpec Translator

Verilog-A

```verilog
// Varactor Model
// Version 1b, 23 October 2006
// Ken Kundert

'include "disciplines.vams"

module varactor(p, n);
    inout p, n;
    electrical p, n;
    parameter real c0 = 1p from (0:inf); // nominal capacitance
    parameter real c1 = 0.5p from (0:c0); // maximum capacitance
    parameter real v0 = 0; // voltage for nominal capacitance
    parameter real v1 = 1 from (0:inf); // voltage for nominal capacitance
    analog begin
        v = V(p, n);
        q = c0*v + c1*v1*ln(cosh((v - v0)/v1));
        i(p, n) <- ddt(q);
    end
endmodule
```

ModSpec (MATLAB)

```matlab
function MOD = yvaractor(yln1qID)

MOD = ee_model();
MOD = add_to_ee_model(MOD, 'modelname', 'varactor');
MOD = add_to_ee_model(MOD, 'terminals', {'p', 'n'});
MOD = add_to_ee_model(MOD, 'explicit_outs', {'i(n)'});
MOD = add_to_ee_model(MOD, 'internal_unks', {});
MOD = add_to_ee_model(MOD, 'parms', {'c0', 1e-12,...
    'c1', 5e-13,...
    'v0', 0,...
    'v1', 1});
MOD = add_to_ee_model(MOD, 'fqi_all', @fqi_all);
MOD = finish_ee_model(MOD);
end

function [fe, qe, fi, qi] = fqi_all(S)
    v2struct(S);
    fe = zeros(1,1);
    qe = zeros(1,1);
    fi = zeros(0,1);
    qi = zeros(0,1);
    v = vpn;
    q = ((c0*v) + (c1*v1)*log(cosh((v - v0)/v1)));
    qi(1) = qe(1) + q;
end
```
Glimpse: Verilog-A → ModSpec Translator

Verilog-A

```verilog
// Varactor Model
// Version 1b, 23 October 2006
// Ken Kundert
// downloaded from the designer's guide (www.designers-guide.org/forums)
// Documentation on the model can be found at www.designers-guide.org/forums

include "disciplines.vams"

module varactor(p, n);
    inout p, n;
    parameter real c0 = 1p from (0:inf); // nominal capacitance
    parameter real c1 = 0.5p from [0:c0]; // maximum capacitance
    parameter real v0 = 0; // voltage for nominal capacitance
    parameter real v1 = 1 from (0:inf); // voltage for capacitance
    real q, v;

    analog begin
        v = V(p, n);
        q = c0*v + c1*v1*ln(cosh((v - v0)/v1));
    end
endmodule
```

ModSpec (MATLAB)

```matlab
function MOD = varactor(ymlID)
    MOD = ee_model();
    MOD = add_to_ee_model (MOD, 'modelname', 'varactor');
    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', {});
    MOD = add_to_ee_model (MOD, 'parms', {'c0', 1e-12, ...,'c1', 5e-13, ...,'v0', 0, ...,'v1', 1});
    MOD = add_to_ee_model (MOD, 'fquid_all', @fquid_all);
    MOD = finish_ee_model(MOD);
end

function [fe, qe, fi, qi] = fquid_all(s)
    y2struct(s);
    fe = zeros(1,1);
    qe = zeros(1,1);
    fi = zeros(0,1);
    qi = zeros(0,1);
    v = vpn;
    q = ((c0*v)+(c1*v1)*log(cosh((v-v0)/v1)));
    qe(1) = qe(1) + q;
end
```
Glimpse: Verilog-A → ModSpec Translator

Verilog-A

```verilog
module varactor(p, n);
  inout p, n;
  parameter real c0 = 1p from (0:inf); // nominal capacitance
  parameter real c1 = 0.5p from [0:c0]; // maximum capacitance
  parameter real v0 = 0; // voltage for nominal capacitance
  parameter real v1 = 1 from (0:inf); // voltage for minimum capacitance

  analog begin
    v = V(p,n);
    q = c0*v + c1*v1*ln(cosh((v - v0)/v1));
    I(p,n) ≜ ddt(q);
  end
endmodule
```

ModSpec (MATLAB)

```matlab
function MOD = varactor(yuniqID)

MOD = ee_model();
MOD = add_to_ee_model(MOD, 'modelname', 'varactor');
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', {});

MOD = add_to_ee_model(MOD, 'parms', {'c0', 1e-12, ...
  'c1', 5e-13, ...
  'v0', 0, ...
  'v1', 1});

MOD = add_to_ee_model(MOD, 'fqei_all', @fqei_all);
MOD = finish_ee_model(MOD);
end

function [fe, qe, fi, qi] = fqei_all(S)
    v2struct(S);
    fe = zeros(1,1);
    qe = zeros(1,1);
    fi = zeros(0,1);
    qi = zeros(0,1);

    v = vpn;
    q = ((c0*v) + (c1*v1)*log(cosh((v-v0)/v1)));
    qe(1) = qe(1) + q;
end
```
MAPP Model Development Flow (3)

Step 3

NEEDS-compatible Verilog-A model (from Step 2)
MAPP Model Development Flow (3)

Step 3

NEEDS-compatible Verilog-A model (from Step 2) → automatic translator → ModSpec Model (C++ API)
MAPP Model Development Flow (3)

Step 3

NEEDS-compatible Verilog-A model (from Step 2)

automatic translator

ModSpec Model (C++ API)

confirm model with DC/AC/TRAN in C++ MAPP

proof of model goodness
MAPP Model Development Flow (3)

Step 3

NEEDS-compatible Verilog-A model (from Step 2) → automatic translator → ModSpec Model (C++ API) → confirm model with DC/AC/TRAN in C++ MAPP → model supported in Open-source Simulators (Xyce) → proof of model goodness
MAPP Model Development Flow (3)

Step 3

NEEDS-compatible Verilog-A model (from Step 2) → automatic translator → ModSpec Model (C++ API) → confirm model with DC/AC/TRAN in C++ MAPP → proof of model goodness → use model in Commercial Simulators → model supported in Open-source Simulators (Xyce)
MAPP Model Development Flow (3)

Step 3

NEEDS-compatible Verilog-A model (from Step 2)

automatic translator

ModSpec Model (C++ API)

“simulation-ready” model deployed

confirm model with DC/AC/TRAN in C++ MAPP

proof of model goodness

use model in Commercial Simulators

model supported in Open-source Simulators (Xyce)
MAPP Model Development Flow (3)

Step 3

NEEDS-compatible Verilog-A model (from Step 2) → automatic translator → use model in Commercial Simulators

“simulation-ready” model deployed

proof of model goodness

ModSpec Model (C++ API)

model supported in Open-source Simulators (Xyce)

model in DC/AC/TRAN in C++ in MAPP

Xyce ModSpec Interface
Glimpse: ModSpec Model in Xyce

ModSpec Model (C++ API)

compile standalone

.so libraries (dynamically loadable)

Xyce-ModSpec Interface

model supported in Xyce
Glimpse: ModSpec Model in Xyce

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

ModSpec Model
(C++ API)

compile standalone

.so libraries
(dynamically loadable)

Xyce-ModSpec Interface

model supported in Xyce

Xyce netlist for inverter
(using MVS ModSpec/C++ model)
Glimpse: ModSpec Model in Xyce

1 *** Test-bench for generating dc response of an inverter
2 3 *** Create 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 33 *** END
34 .end

Xyce netlist for inverter (using MVS ModSpec/C++ model)
Glimpse: ModSpec Model in Xyce

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

ModSpec Model (C++ API)

- compile standalone
- .so libraries (dynamically loadable)

Xyce-ModSpec Interface

- model supported in Xyce

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

- model's name
- model parameter: name of .so library

A. Gokcen Mahmutoglu, amahmutoglu@berkeley.edu
**Glimpse: ModSpec Model in Xyce**

1. *** Test-bench for generating dc response of an inverter
2. *** Create sub-circuit for the inverter
3. `.subckt` inverter Vin Vout Vvdd Vgnd
4. `yModSpec_Device X1 Vvdd Vin Vout Vvdd MVSmod type=-1 W=1.0e-4`  
   `Lqdr=32e-7 dLq=8e-7 Cg=2.57e-6 beta=1.85 alpha=3.5 Tjun=300`  
   `Cif = 1.38e-12 Cof=1.47e-12 phib=1.2 gamma=0.1 mc=0.2`  
   `CTM_select=1 Rs0=100 Rd0 = 100 n0=1.68 nd=0.1 vxo=7542204`  
   `mu=165 Vt0=0.5535 delta=0.15`
5. `yModSpec_Device X0 Vout Vin Vgnd Vgnd MVSmod type=1 W=1e-4`  
   `Lqdr=32e-7 dLq=9e-7 Cg=2.57e-6 beta=1.85 alpha=3.5 Tjun=300`  
   `Cif=1.38e-12 Cof=1.47e-12 phib=1.2 gamma=0.1 mc=0.2`  
   `CTM_select=1 Rs0=100 Rd0=100 n0=1.68 nd=0.1 vxo=1.2e7`  
   `mu=200 Vt0=0.4 delta=0.15`
6. `.model` MVSmod MODSPEC_DEVICE SONAME=MVS_ModSpec_Element.so;
7. `.model` line
8. `.ends`
9. *** circuit layout
10. `Vsup sup 0 1`
11. `Vin in 0 0`
12. `Vsource source 0 0`
13. `X2 in out sup 0 inverter`
14. *** simulation
15. `.dc` Vin 0 1 0.01
16. `.print` dc V(in)
17. *** END
18. `.end`

---

**ModSpec Model (C++ API)**

- Compile standalone
- `.so` libraries (dynamically loadable)
- Xyce-ModSpec Interface
- Model supported in Xyce
- DC sweep using Xyce
- Xyce netlist for inverter (using MVS ModSpec/C++ model)
Glimpse: ModSpec models in Xyce
Glimpse: ModSpec models in Xyce

[Diagram of a circuit with nodes and components labeled e1, e2, e3, e4, e5, and Vdd.]

[Graph showing time-domain waveforms for signals e1 to e5.]
Glimpse: ModSpec models in Xyce

20 ModSpec BJTs
Glimpse: ModSpec models in Xyce

Xyce-ModSpec-Interface supports voltage limiting (e.g., pnjlim, fetlim, limvds)

20 ModSpec BJTs
Glimpse: ModSpec models in Xyce

Xyce-ModSpec-Interface supports voltage limiting (e.g., pnjlim, fetlim, limvds)

20 ModSpec BJTs

multiple ModSpec models in one netlist
Glimpse: ModSpec models in Xyce

Xyce-ModSpec-Interface supports voltage limiting (e.g., pnjlim, fetlim, limvds)

20 ModSpec BJTs

multiple ModSpec models in one netlist runs in other analyses as well, e.g., HB.
MAPP: Features

- Works entirely in MATLAB
- Help system (start with `help MAPP`)
- Automatic differentiation (`vecvalder`)
- Executable device specification (`ModSpec`)
- DC, AC, transient analyses
  - also noise, homotopy, HB, shooting, PPV, MOR, etc. (not released yet)
- Automated testing system exercising suite of tests
- Verilog-A model translator (`VAPP`)
- Xyce – ModSpec interface
Summary

http://MAPP.eecs.berkeley.edu