International Conference
MIXED DESIGN
MIXDES 2006
Gdynia, POLAND
22 - 24 June 2006

ADVANCED DEVICE MODELING USING AUTOMATIC DIFFERENTIATION IN A MIXED DOMAIN CIRCUIT SIMULATOR

M. B. STEER, N. M. KRIPLANI, W. JANG
NORTH CAROLINA STATE UNIVERSITY, U.S.A
A. MANTOOTH
UNIVERSITY OF ARKANSAS, U.S.A

KEYWORDS: automatic differentiation, EKV model, transient circuit simulation

ABSTRACT: The advent of automatic differentiation, the adoption of object-oriented implementation paradigms, and compiler-support for function overloading enables relatively straightforward implementation of device models with typically 10% of the code required for standard device implementations. In this paper implementation of the EKV model in the *f*REEDA™ circuit simulator is reported. *f*REEDA™ is capable of transient, wavelet, harmonic balance, large-signal transient noise, DC and AC analyses of circuits using a single device model implementation. Automatic differentiation enables the calculation of device derivatives with analytic accuracy but without errors common in manual development of device derivatives. *f*REEDA™ achieves unconditional convergence through the use of parameterized state variables that obviates the end for local convergence control or homotopy techniques. A dynamic range in transient simulation exceeding 160 dB has been achieved.

INTRODUCTION

The *f*REEDA™ circuit simulator has been a long-term project that represents a fundamental new capability in circuit simulation [1–8]. Many new paradigms have been tested and incorporated. As *f*REEDA™ is open source software with a GNU Public License (or GPL) it can serve as a vehicle to embed circuit simulation research in a general purpose simulator framework. The commitment to GPL has enabled mathematical advances coded in GPL software to be exploited. A commitment to using the C++ programming language and rigorous adherence to object-orientation has enabled advances by a relatively small group of developers. This paper focuses on the capabilities of *f*REEDA™ in providing convenient implementation of the EPFL-EKV MOSFET model. The EPFL-EKV MOSFET model is a scalable and compact simulation model built on fundamental physical properties of the MOS structure [9,10]. Compared to most other device models for circuit simulators, the EKV MOSFET model is well documented [9]. Because of this it was relatively easy to implement the model in *f*REEDA™ .

SIMULATOR STRUCTURE

Circuits can be described using graph theory. Graphs graphically describe the connectivity of a circuit and over time the tradeoff of many concepts has led to several systematic approaches to circuit topology. We will be very explicit about the use of the terms node and terminal in the discussion that follows. *Terminal* is also the preferred term in VHDL. We describe two ways of looking at circuits and call these the *Circuit View* (used in Spice) and the *Network View* (used in *f*REEDA™). These views differ in the ways we assign nodes and edges to terminals and elements. In the Circuit View nodes are always terminals and, also, edges denote elements. In the Network

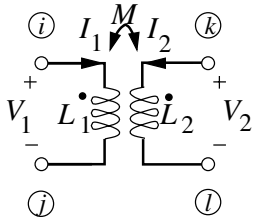

Fig. 1. A transformer represented as a two-port.

View both terminals and elements are nodes and edges describe connectivity and have no other attributes. When the term node is used it must be assumed that we could be talking about a terminal or an element. The network view is used in *f*REEDA™ and greatly facilitates the incorporation of multi-terminal, multi-physics elements. Circuit theory has evolved to include a common reference terminal to which all voltages in a circuit are referred. However it is generally not feasible to define nodal voltages, or a single reference point, in a spatially distributed system. An example of where classical circuit concepts break down is seen with respect to using the transformer shown in Fig. 1. In Spice a circuit with a transformer element would be handled by either shorting the two reference terminals together or else using large resistance to connect the terminals to ensure a connected graph. Problems can be seen in handling the cascode transformer circuit of Fig. 2. Fig. 2(a) is a disconnected graph which is rendered a connected graph in Fig. 2(b) by shorting the two reference terminals of each transformer. This connection would clearly give erroneous results. In the *f*REEDA™ paradigm a local reference terminal introduced in addition to the global reference terminal (which is also a local reference terminal) so that the disconnected graph can be correctly handled without introducing additional circuit elements or connections.

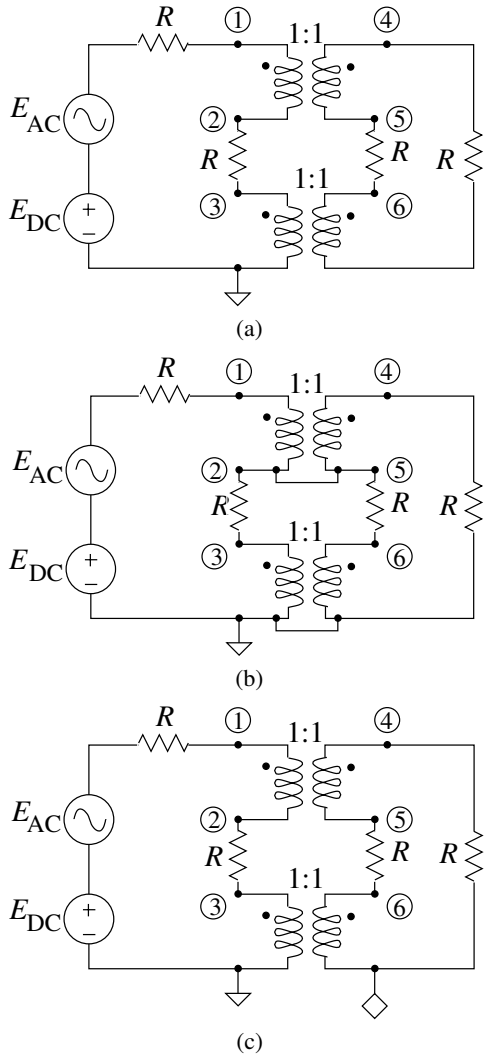

Fig. 2. Representations of interconnected two-ports: (a) two transformers in cascode; (b) the circuit with shorted lower windings of each transformer; and (c) circuit with one global reference terminal and an additional local reference terminal.

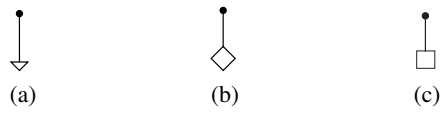

Fig. 3. Reference terminals: (a) conventional global reference terminal; (b) local reference terminal; and (c) element reference terminal.

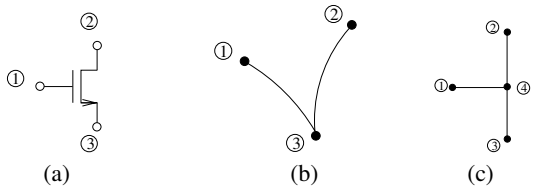

Fig. 4. Graph representations of a three-terminal element: (a) a mosfet transistor as a three-terminal element; (b) circuit graph of the three-terminal element; and (c) network graph of the three-terminal element.

*f*REEDA™ supports three types of reference terminals, see Fig. 3. The global reference, Fig. 3(a), is the ground of conventional circuit theory. The LRT, Fig. 3(b), is a generalization of ground with different segments of a circuit locally referenced to LRT. The global ground is just a special case of an LRT. Circuit elements also have their own reference, see Fig. 3(c), and this looks like an LRT until the circuit element is connected. The number of separate parts of a circuit is the number of LRTs. However all of the reference, global and local, have exactly the same significance: they are all local reference terminals.

*f*REEDA™ uses a network graph view of a circuit rather than the conventional circuit graph. The two views are contrasted in Fig. 4 for a three-terminal element. From the perspective of relating to the physical behavior of multi-terminal elements, the network graph is convenient as edges do not have properties, nodes do, and nodes can have elaborate properties.

The *f*REEDA™ analysis process treats every element the same way and develops the circuit equations at the top of the analysis hierarchy. This model is not used in Spice as it requires element-by-element local convergence control. The circuit equations are created element-by-element. One of the effects of this is that each element in Spice is in fact largely hand-crafted and there is little modeling uniformity. One important concept that enables the *f*REEDA™ approach is parameterization in which strong nonlinearities of say $=$ $(\ )$ are replaced by two separate moderately nonlinear functions: $=$ $(\ )$ and $=$ $(\ )$, but still $=$ $(\ )$. Now the iterative solver works on . Parameterization requires a different error formulation composed of KCL and KVL-like errors, or fortuitously, an energy norm error. This enables circuits with little current to be solved robustly.

Key concepts behind *f*REEDA™ are:

1. The use of local reference terminals rather than a single global ground.
2. The incorporation of mixed domains, such as thermal and mechanical models, into a seamless global simulator.
3. Rapid implementation of new device models.
4. Unified device modeling. Models that can be coded once and be used in many different types of simulation analyses.
5. Simulator architecture that enables best-practice numerical approaches to be utilized.
6. Object-oriented modeling and programming practice.
7. Uncompromising commitment to accuracy throughout the simulator.

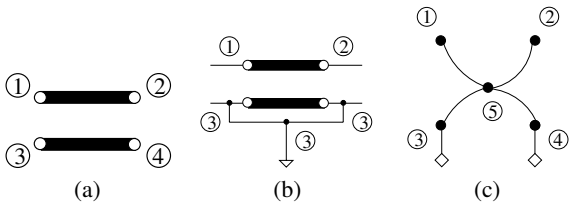

Fig. 5. Representation of a transmission line: (a) four terminal element; (b) as a two-port; and (c) network graph representation.

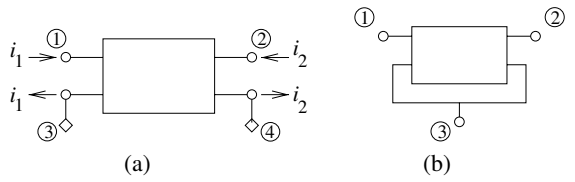

Fig. 6. Two ports: (a) with a local reference terminals (LRTs); and (b) with a global reference terminal (GRT).

Another example of using the local reference terminal concept is in modeling of spatially distributed circuit, the simplest of which is a transmission line. The transmission line is a two-port, four-terminal element as shown in Fig. 5(a). A transmission line, as with any two port, has the special property that the currents entering the terminals of a port sum to zero. With a two-terminal port the currents are opposite and equal. With a transmission line this is a consequence of energy minimization when two conductors of the transmission line are in close proximity. This property is enforced in the circuit graph representation, Fig. 5(b) with each port represented by a single edge. Thus the currents at the terminals of a port are forced to be opposite and equal. This balancing does not naturally occur with the network graph of Fig. 5(c). Instead the concept of local reference groups (LRGs) is used. Each LRG has a single local reference terminal (LRT) and one or more other terminals. Kirchhoff's current law applies to the LRT. Thus there is a fundamental difference between the two port representation with LRTs, Fig. 6(a), and the (incorrect) representation of a two port with a global reference terminal (GRT), Fig. 6(b). Using the conventional formulation of the nodal admittance matrix, specific relations between the terminal voltages of two LRGs are erroneously imposed. In fact the voltages of different LRTs must be allowed to float with respect to each other and the general KCL condition applies to individual LRGs and not to the whole circuit.

With a single (but incorrect) GRT, KCL applied to the reference terminal results in one equation, i.e. from Fig. 6(b):

$$_1 + \ _3 + \ _2 + \ _4 = 0 \qquad (1)$$

However with the LRTs, KCL applied to the reference terminals results in two equations, i.e. from Fig. 6(a):

$$\begin{aligned} _1 + \ _3 &= 0, \\ _2 + \ _4 &= 0 \end{aligned} \qquad (2)$$

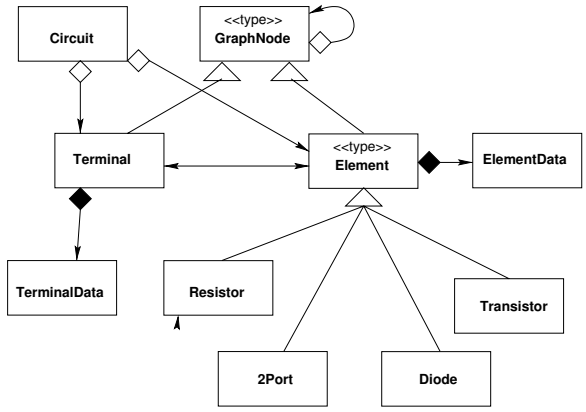

Fig. 7. Universal Markup Language (UML) diagram of an object-oriented implementation of the network graph.

These are subtly different constraints, but can result in fundamentally different behaviors of circuits. While we see that it is necessary to use the LRTs and LRGs with the network graph, the concepts can equally be used with circuit graphs.

In object-oriented (OO) thinking (and OO programming), the network view is a much more convenient way of representing and thinking about a circuit. The circuit graph is quite satisfactory for two-terminal elements, an edge has two-terminals, but it is not as convenient for representing multi-terminal elements. It is difficult to expand the simulator to handle additional elements and very difficult to make structural changes, say add another analysis type to the those envisioned when the simulator was designed. In the Network View the Network Graph can be represented quite easily as can be seen in the UML diagram for a network graph shown in Fig. 7. This UML diagram is implemented in *f*REEDA™ . The architecture of *f*REEDA™ is described by many UMLs and their development took one year before one line of code was written.

Rigorous adherence to precise topology and accuracy has resulted in transient simulation in *f*REEDA™ having a very high dynamic range. One of the key developments was improving the estimation of the truncation error in moving from time-point to time-point. The conventional approach is to compare the nonlinear solution at a time-point to a forward Euler (straight-line) extrapolation from the solution at the previous time-point. This clearly does a poor job of estimating the error when the signal is changing in anything other than a straight line. The net effect is that time points are much too close and there is accumulated numerical error. The *f*REEDA™ approach is to use two different iterative solutions, here backward Euler and Trapezoidal and compare them to get a better estimate of error. Here a two-tone test was used in which one of the tones was held constant and the other varied. The fidelity of the third-order intermod was used as a criterion. *f*REEDA™ achieves a dynamic range exceeding 160 dB in transient circuit simulation. Better than what can be achieved in harmonic balance analysis.

EKV MODEL IMPLEMENTATION

The implementation of the EKV model is an example of how the features of *f*REEDA™ come together to make model develop quite simple. In the Appendix the full code (almost) of the EKV model is presented. This is dramatically simpler than in a Spice-like implementation. The code includes a class setting up the element information (Ekv:einfo), a constructor (Ekv::Ekv) and an initialization routine (Ekv::init). There is no specific coding of derivatives. The manual derivation of the formulas to calculate the derivatives and the coding of these derivatives is an error prone step and a major hurdle to model implementation in traditional circuit simulators. The derivatives are calculated using automatic differentiation which is implemented at the top of the simulator hierarchy. There are two evaluation routines: eval1 and eval1. The precise way these are used and which derivatives are calculated depends on the analysis type being used. Function overloading enables the single model code to be used in any of a large number of analyses. Altogether *f*REEDA™ has 32 analysis types and the model code has never had to be rewritten for new analysis types. Even with a recent major change on the libraries used by *f*REEDA™ , consolidating the number of libraries used from 8 to 3, no change to the model code has been required. In transient analysis using associated discrete modeling eval1 is called and then eval2 at each time/newton iteration. The routine eval1 has as its input the state variables, the 's. The output of the evaluation routine are the voltages at the terminals, [3]- [5], the DC currents at the terminals, [0]- [2], and the charges 1[0]- 1[2]. The routine eval2 has as its input the voltages, [3]- [5], and the time derivatives of the charges 1[0]– 1[2] (1 = 1). These derivative are calculated in the transient analysis routine using automatic differentiation. This *f*REEDA™ code is straightforward and can be automatically generated using the Paragon™ model generation computer program using a hardware description language input script [11].
The output of the model is fully compliant with the documentation. As an example the netlist of an inverter with a 1 GHz pulse source is shown in Figure 8. Running this yielded the output shown in Figure 9.

CONCLUSION

*f*REEDA™ is open source software with a GNU Public License (or GPL) and it is hoped that it will serve as a vehicle to enable circuit simulation research to ne incorporated in a general purpose simulator. It should be the simulator of choice with a large body of supported models. *f*REEDA™ is written in the C++ programming language and the structure rigorously adheres to object-oriented principals.

```
.tran2 tstop=10e-9 tstep=10e-12
ekv: m1 30 20 10 10 l=1e-6 w=20e-6
+    type=-1
ekv: m2 30 20 0 0 l=1e-6 w=20e-6
vpulse:vgate 20 0  v1=0 v2=3
+    pw=0.5e-9 per=1e-9
vsource:vs 10 0 vdc = 3.0
.out plot term 20 vt in "pulse.in"
.out plot term 30 vt in "pulse.out"
.end
```

Fig. 8. Inverter netlist in native fREEDA™ format using the EKV2.6 model.

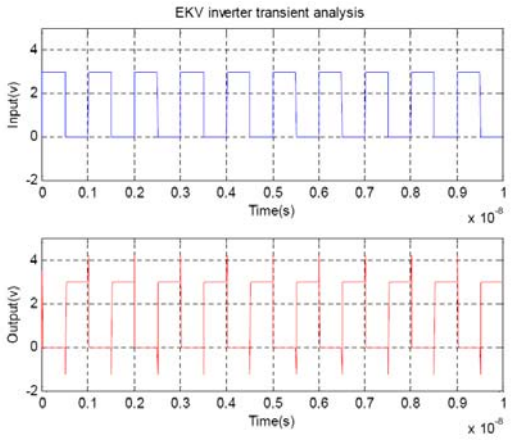

Fig. 9. Output of the EKV inverter simulation.

THE AUTHORS

Michael Steer, Nikhil Kriplani and Wonhoon Jang are with the Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, North Carolina 27695, U.S.A
Alan Mantooth is with the Department of Electrical and Computer Engineering, University of Arkansas, Fayetteville, Arkansas 72701, U.S.A.
E-Mail: m.b.steer@ieee.org

ACKNOWLEDGEMENT

This work was supported by the U.S. Army Research Office as a Multi-disciplinary University Research Initiative on Standoff Inverse Analysis and Manipulation of Electronic Systems under grant number W911NF-05-1-0337. We gratefully acknowledge discussions with C. E. Christofferson and S. Luniya

REFERENCES

[1] S. Luniya, M. B. Steer and C. Christoffersen, "High Dynamic Range Transient Simulation of Microwave Circuits," *IEEE Radio and Wireless Conference*, p. 487, 19–22 Sep. 2004.

[2] N. M. Kriplani, A. Victor and M. B. Steer, “Time-domain modelling of phase noise in an oscillator,” *36 th European Microwave Conference,* Sept. 2006, In Press.

[3] C. E. Christoffersen, U. A. Mughal, and M. B. Steer, ”Object oriented microwave circuit simulation,” Int. J. on RF and Microwave Computer Aided Engineering, Vol. 10, Issue 3, May/June 2000, pp. 164–182.

[4] C. E. Christoffersen and M. B. Steer, “Implementation of the local reference node concept for spatially distributed circuits,” *Int. J. on RF and Microwave Computer Aided Engineering,* Vol. 9, No. 5, Sept. 1999, pp. 376–384.

[5] M. B. Steer, J. F. Harvey, J. W. Mink, M. N. Abdulla, C. E. Christoffersen, H. M. Gutierrez, P. L. Heron, C. W. Hicks, A. I. Khalil, U. A. Mughal, S. Nakazawa, T. W. Nuteson, J. Patwardhan, S. G. Skaggs, M. A. Summers, S. Wang, and A. B. Yakovlev, “Global modeling of spatially distributed microwave and millimeter-wave systems,” *IEEE Trans. Microwave Theory Techniques,* June 1999, pp. 830–839.

[6] S. Luniya, M. B. Steer and C. E. Christoffersen, “High dynamic range transient simulation of microwave circuits,” *IEEE Radio and Wireless Conf. (RAWCON),* Sept. 2004.

[7] F. P. Hart, N. Kriplani, S. R. Luniya, C. E. Christoffersen and M. B. Steer, “Streamlined Circuit and Device Model Development with fREEDA and ADOL-C,” in *Automatic Differentiation: Applications, Theory, and Implementations,* edited by H. M. Bücker, G. F. Corliss, P. Hovland, U. Naumann and B. Norris, Series: Lecture Notes in Computational Science and Engineering, Vol. 50, Springer, New York, NY, 2005.

[8] J. Ding, D. Linton and M. B. Steer, “Compact Electro-thermal Modelling and Simulation of InP HBT based on the Local Reference Concept,” *36 th European Microwave Conference,* Sept. 2006.

[9] M. Bucher, C. Lallement, C. Enz, F. Theodoloz and F. Krummenacher, “The EPFL-EKV MOSFET Model Equation for Simulation,” Technical Report, Electronics Laboratories, Swiss Federal Institute of Technology(EPFL), Lausanne, Switzerland, Model Version 2.6, June 1997.

[10] C. C. Enz, F. Krummenacher and E. A. Vittoz, “An Analytical MOS Transistor Model Valid in All Region of Operation and Dedicated to Low-Voltage and Low Current Applications,” *J. Analog Integrated Circuits and Signal Processing*, Vol. 8, pp. 83–114, 1995.

[11] A. S. Kashyap, C. Vemulapally and H. A. Mantooth, “VHDL - AMS modeling of silicon carbide power semiconductor devices,” *Proceedings 2004 IEEE Workshop on Computers in Power Electronics,* Aug. 2004, pp. 50–54.

APPENDIX: EKV MODEL CODE

Complete code of the EPFL-EKV MOSFET model (version 2.6) implementing both the n and p type devices.

```
#include "../network/CircuitManager.h"
#include "../network/Element.h"
#include "../network/NAdolcElement.h"
#include "Ekv.h"
#include "../analysis/TimeDomainSV.h"

// Static members
const unsigned Ekv::n_par = 44;

// Element information
ItemInfo Ekv::einfo = {
  "ekv",
  "EPFL EKV MOSFET model",
  "Wonhoon Jang",
  DEFAULT_ADDRESS"elements/Ekv.html" };

// Parameter information
ParmInfo Ekv::pinfo[] = {
  {"type", "N-channel or P-channel MOS", TR_DOUBLE, false},
// Device input variables
  {"l", "Gate length (m)", TR_DOUBLE, false},
  {"w", "Gate width (m)", TR_DOUBLE, false},
  {"np", "Parallel multiple device number", TR_DOUBLE, false},
  {"ns", "Serial multiple device number", TR_DOUBLE, false},
.
. (PARAMETERS DELETED FOR BREVITY)
.
  {"tmp", "Model simulation temperature (K)", TR_DOUBLE, false},
};

Ekv::Ekv(const string& iname)
  : NAdolcElement(&einfo, pinfo, n_par, iname)
{
  // Set default parameter values
  paramvalue[0]  = &(type = NMOS);
  paramvalue[1]  = &(l = 1e-6);
  .
  . PARAMETERS DELETED FOR TEH SAKE OF BREVITY
  .
  paramvalue[43] = &(tmp = 300.15);

  // Set the number of terminals
  setNumTerms(4);

  // Set flags
  setFlags(NONLINEAR | ONE_REF | TR_TIME_DOMAIN);

  // Set number of states
  setNumberOfStates(3);

  // Set number of levels, sets the number of derivatives to take.
  // In this model, the voltage derivatives are not used, but the
  // charge values are calculated from the current node voltages and
  // then the derivative of the charge is used to determine the
  // AC current.  Thus, starting with the first level of the input
  // voltages, this model needs two tape levels to generate the output
  // current AC + DC.
  setNlevels(2);
}

void Ekv::init() throw(string&)
{
  // Set Constants
  epsilonox = scale*34.5e-12;
  epsilonsi = scale*104.5e-12;
  q=1.602e-19;
  k=1.3807e-23;
  Tref=300.15;
  vtT = (k*tmp)/q; // Thermal voltage
  vtTnom = (k*tnom)/q;
  vtTref = (k*Tref)/q;
  egT = 1.16-0.000702*tmp*tmp/(tmp+1108); // Energy gap
  egTnom = 1.16-0.000702*tnom*tnom/(tnom+1108);
  egTref = 1.16-0.000702*Tref*Tref/(Tref+1108);
  niT = 1.45e16*(tmp/Tref)*exp(egTref/(2*vtTref)-egT/(2*vtT));
  niTnom = 1.45e16*(tnom/Tref)*exp(egTref/(2*vtTref)-egTnom/(2*vtTnom));

  // For P-channel devices
  if(type == -1){
    tcv = type * tcv;
    vto = type * vto;  }

  // Calculate any missing parameters from user-defined settings
  // COX
  if(cox == 0.0){
    if (tox > 0.0)
      cox = epsilonox / tox;
    else  cox = 0.7e-3; }
  // GAMMA
  if(gamma == 0.0){
    if (nsub > 0.0)
      gamma = sqrt(2.0 * epsilonsi * nsub * 1e6) / cox;
    else  gamma = 1; }
  // PHI
  if(phi == 0.0){
    if (nsub > 0.0)
      phi = 2.0 * vtTnom * log(nsub * 1e6 / niTnom);
    else  phi = 0.7; }
  // VTO
  if(vto == 0.0){
    if(vfb != -2003)
      vto = vfb + phi + gamma * sqrt(phi);
    else  vto = 0.5; }
  // KP
  if(kp == 0.0){
    if(uo > 0.0)
      kp = uo * cox * 1e-4 /*(m^2/cm^2)*/;
    else  kp = 5.0e-5;  }
  // UCRIT
  if(ucrit == 0.0){
    if((vmax > 0) && (uo > 0))
      ucrit = vmax / (uo * 1e-4);
```

```
    else  ucrit = 2.0e6;  }
  // EO
  if(eo == 0.0){
    if(theta >=0)
      eo = 0.0;
    else   eo = 1e12;  }

  // Intrinsic parameters temperature dependence
  vtoT=vto-tcv*(tmp-tnom);
  kpT=kp*pow(tmp/tnom,bex);
  ucritT=ucrit*pow(tmp/tnom,ucex);
  phiT=phi*tmp/tnom-3*vtT*log(tmp/tnom)-egTnom*tmp/tnom+egT;
  ibbT=ibb*(1+ibbt*(tmp-tnom));

  // create tape
  IntVector var(3);
  var[0] = 0;
  var[1] = 1;
  var[2] = 2;
  IntVector novar;
  DoubleVector nodelay;
  createTape1(var, novar, nodelay, 3, 6);
  createTape2(0,6);
}

void Ekv::eval1(adoublev& x, adoublev& xt, adoublev& y1, adoublev& z1)
{
  // x[0]: vds   x[1]: vgs   x[2]: vbs
  // x[3]: dvds/dt   x[4]: dvgs/dt    x[5]: dvbs/dt
  // z1[3]: vdb , z2[3]: id
  // z1[4]: vgb , z2[4]: ig
  // z1[5]: vsb , z2[5]: is
  double weff = w + dw;
  double leff = l + dl;
  double vtoa = vtoT + avto / sqrt(np * weff * ns * leff);
  double kpa = kpT * (1 + akp / sqrt(np * weff * ns * leff));
  double gammaa = gamma + agamma / sqrt(np * weff * ns * leff);
  double cEps = 4 * pow(22e-3,2);
  double cA = 0.028;
  double xi  = cA * (10 * leff / lk -1);
  double deltavRSCE = 2 * qo / (cox * pow(1 + 0.5 * (xi + sqrt(xi*xi + cEps)),2));
  double vc = ucritT * ns * leff;
  double lc = sqrt(epsilonsi * xj / cox);
  double lmin = ns * leff / l0;
  if(type == 1)
    eta = 0.5;
  else  eta = 0.3333333333333;
  double qbo = gammaa * sqrt(phiT);
  double qox = 0;
  double C_ox = cox * np * weff * ns * leff;

  //All the active variables for ADOL-C "adoubles" must be initialiazed here
  adouble vgprime, vpol, vpo, vsprime, vdprime, gammao, gammaprime, vpl, vp;
  adouble n, i_f, vdss, vdssprime, deltav, vds, vip, deltal, lprime, leq;
  adouble irprime, ir, betao, betaoprime, beta, is, ids, vib; //vpprime;
  adouble idb1, idb, id, nq, xf, xr, qd, qs, qi, qb1, qb, qg, QI, QB, QD;
  adouble QS, QG;

  // Effective gate voltage in cluding reverse short channel effect
  vgprime =  type*x[1] - vtoa - deltavRSCE + phiT + gammaa * sqrt(phiT);

  // Effective substrate factor including charge-sharing for short and narrow
  // channels
  // Pinch-off voltage for narrow-channel effect
  vpol = vgprime - phiT - gammaa * (sqrt(vgprime + gammaa*gammaa / 4) - gammaa
  / 2);
  condassign(vpo, vgprime, vpol, -phiT);

  // Effective substrate factor accounting for charge-sharing
  vsprime=0.5*(type*x[2]+phiT+sqrt(pow(type*x[2]+phiT,2) + 16 * vtT*vtT));
  vdprime=0.5*( type*x[0]+phiT+sqrt(pow( type*x[0]+phiT,2) + 16 * vtT*vtT));

  // Pinch-off voltage including short- and narrow-channel effect
  gammao = gammaa - epsilonsi * (leta * (sqrt(vsprime) + sqrt(vdprime))
  / leff - 3 * weta * sqrt(vpo + phiT) / weff) / cox;
  gammaprime = 0.5 * (gammao + sqrt(gammao*gammao + 0.1 * vtT));
  vpl = vgprime - phiT - gammaprime * (sqrt(vgprime+pow(gammaprime / 2,2)) -
  gammaprime / 2);
  condassign(vp, vgprime, vpl, -phiT);

  // Slop factor
  n = 1 + gammaa / (2 * sqrt(vp + phiT + 4 * vtT));

  // Forward normalized current
  i_f=log(1+exp((vp-type*x[2])/(2*vtT)))*log(1+exp((vp-type*x[2])/(2*vtT)));

  // Velocity saturation voltage
  vdss = vc * (sqrt(0.25 + vtT * sqrt(i_f) / vc) - 0.5);

  // Drain-to-source saturation voltage for reverse normalized current
  vdssprime = vc * (sqrt(0.25 + vtT * (sqrt(i_f) - 0.75 * log(i_f))/vc) - 0.5) +
  vtT * (log(vc / (2 * vtT)) - 0.6);

  // Channel-length modulation
  deltav = 4 * vtT * sqrt(lambda * (sqrt(i_f) - vdss / vtT) + 1 /64);
  vds = ( type*x[0] - type*x[2]) / 2;
  vip = sqrt(vdss*vdss + deltav*deltav) - sqrt(pow(vds - vdss,2) + deltav*deltav);
  deltal = lambda * lc * log(1 + (vds - vip) / (lc * ucritT));

  // Equivalent channel length including channel-length moculation and velocity
  // saturation
  lprime = ns * leff - deltal + (vds + vip) / ucritT;
  leq = 0.5 * (lprime + sqrt(lprime*lprime + lmin*lmin));

  // Reverse normalized current
  irprime = log(1 + exp(((vp - vds - type*x[2] - sqrt(vdssprime*vdssprime +
  deltav*deltav) + sqrt((vds-vdssprime)*(vds-vdssprime) + deltav*deltav)) /
  vtT)/ 2))*log(1 + exp(((vp - vds - type*x[2] - sqrt(vdssprime*vdssprime +
  deltav*deltav) + sqrt((vds-vdssprime)*(vds-vdssprime) + deltav*deltav)) / vtT)/ 2));

  // Reverse normalized currect for mobility model, intrinsic
  //charges/capacitances, thermal noise model and NQS time-constant
  ir=log(1+exp((vp-type*x[0])/(2*vtT)))*log(1+exp((vp-type*x[0])/(2*vtT)));

  // Transconductance factor and mobility reduction due to vertical field
  betao = kpa * np * weff / leq;
  betaoprime = betao * (1 + cox * qbo / (eo * epsilonsi));

  // Simple mobility reduction model
  //vpprime = 0.5 * (vp + sqrt(vp*vp + 2* vtT*vtT));
  //beta = betao / (1 + theta * vpprime);

  // Quasi-static model equations
  // Dynamic model for the intrinsic node charges
  nq = 1 + gammaa / (2 * sqrt(vp + phiT +1e-6));

  // Normalized intrinsic node charges
  xf = sqrt(0.25 + i_f);
  xr = sqrt(0.25 + ir);
  qd = -nq * (4 * (3 * xr*xr*xr + 6 * xr*xr * xf + 4 * xr * xf*xf + 2 * xf*xf*xf)
       / (15 * pow(xf + xr,2)) - 0.5);
  qs = -nq * (4 * (3 * xf*xf*xf + 6 * xf*xf * xr + 4 * xf * xr*xr + 2 * xr*xr*xr)
       / (15 * pow(xf + xr,2)) - 0.5);
  qi = qs + qd;
  qb1 = -gammaa * sqrt(vp + phiT + 1e-6) / vtT - (nq - 1) * qi / nq;
  condassign(qb, vgprime, qb1, -vgprime / vtT);
  qg = -qi - qox - qb;

  // Rigorous mobility reduction model
  beta = betaoprime / (1 + cox * vtT * fabs(qb + eta*qi) / (eo * epsilonsi));

  // Specific current
  is = 2 * n * beta * vtT*vtT;

  // Drain-to-source current
  ids = type*is * (i_f - irprime);

  // Impact ionization current
  vib =  type*x[0] - type*x[2] - 2 * ibn * vdss;
  idb1 = ids * iba * vib * exp(-ibbT * lc / vib) / ibbT;
  condassign(idb, vib, idb1, 0);
  id = ids + idb;

  // Total charges
  QI = C_ox * vtT * qi;
  QB = C_ox * vtT * qb;
  QD = C_ox * vtT * qd;
  QS = C_ox * vtT * qs;
  QG = C_ox * vtT * qg;

  y1[0] =  type*QD;
  y1[1] =  type*QG;
  y1[2] =  type*QS;

  // Assign DC currents
  z1[0] =  id;    //DC Drain current
  z1[1] = 0.0;  //DC Gate current
  z1[2] = -id;  //DC Source current

  // Assign known output voltages
  z1[3] = x[0];  // vdb
  z1[4] = x[1];  // vgb
  z1[5] = x[2];  // vsb
}

void Ekv::eval2(adoublev& dy1, adoublev& z1, adoublev& y2, adoublev& z2)
{
  z2[0] = z1[3]; //Vdb
  z2[1] = z1[4]; //Vgb
  z2[2] = z1[5]; //Vsb

  z2[3] = z1[0] + dy1[0]; //Drain current  Drain charge derivative
  z2[4] = z1[1] + dy1[1]; //Gate current, gate charge derivative
  z2[5] = z1[2] - dy1[2]; //Source current, source charge derivative
}
```