CONTROLLING TCAD APPLICATIONS WITH AN OBJECT-ORIENTED DYNAMIC DATABASE

Robert Klima, Tibor Grasser, and Siegfried Selberherr
Institute for Microelectronics
TU Wien
Gußhausstraße 27–29,
Vienna, Austria
E-mail: klima@iue.tuwien.ac.at

KEYWORDS

Object-oriented, Control systems, Computer Aided Engineering, Intelligent Simulation Environments, Electronics.

ABSTRACT

State-of-the-art TCAD applications like device and process simulators require a huge number of different information in addition to the device input data. Simulation parameters and parameter dependencies, models to use, material information, or circuit descriptions must be handled in an efficient and comfortable manner. Simple input deck files which are just a sequence of static keywords or the use of the command line are no longer sufficient. To obtain a maximum of flexibility we use a specialized object-oriented dynamic database to control the simulators.

INTRODUCTION

To perform a specific simulation task TCAD applications like the complex circuit- and device-simulator MINIMOS-NT(Binder et al. 1998) require a huge amount of information. This information falls into different categories and covers for instance simulation modes, iteration schemes, models, model parameters, material characteristics, input and output parameters, or circuit descriptions.

Conventional input files are a sequence of pure static parameters. The parameter values are read or computed while reading the file. All dependencies between the parameters are lost because only the values are stored. Default values are normally stored internally in the application and are not accessible to the user. So the user can neither inquire the default values nor alter them. Due to the different nature of the information conventional simulators often use different input languages. The disadvantage of this approach is that different input modules are needed to handle information providing different functionalities. Different modules make maintenance more difficult. Moreover, the user has to learn different input languages.

We present an object-oriented dynamic database (Klima et al. 2000) which was developed to meet the requirements of state-of-the-art TCAD applications.

Requirements for the Input Deck database are:

- Variables store values or formulas to describe dependencies. When the value of a variable is changed, depending variables are recomputed.
- Application specific functions can be integrated to consistently enlarge the functionality of the Input Deck database.
- All necessary kinds of different information, even circuits, can be described.
- Information can be grouped and organized hierarchically.
- The user has access to default settings
- The application can query stored information, modify the database, and store runtime information.

To demonstrate the capabilities of the Input Deck database and how it can be used to control TCAD applications we describe the material handling and an example circuit simulation MINIMOS-NT.

INFORMATION MANAGEMENT

The Input Deck database has its own description language, the Input Deck programming language (IPL). The syntax of the IPL is similar to that of the C++ programming language but it is not a sequential programming language as it describes the entries in the database. Items stored in the database may be grouped hierarchically. The most important items are variables and sections.

Variables may contain values or expressions which are evaluated by the Input Deck evaluation module (see Section) which is part of the Input Deck database. In contrast to the handling of common sequential programming languages, variables can be seen as states depending on other variables (states). The dependence is described by the formula assigned.

Sections are named containers holding an arbitrary set of variables and an arbitrary number of nested subsections. Starting at the *root section* denoted by the tilde (‘~’) a section tree is formed.

Inheritance is a powerful feature of the Input Deck database. Inheritance can be used to pass a complex tree of subsections and keywords to a new section creating a derived section. Since this inheritance is implemented dynamically all changes in a base section are transparent to the derived section. This feature is necessary for instance to build up the material database in MINIMOS-NT (see Section).

```
Base {
   a = 1;
   b = a + 1;
}
Extended : Base {
   a = 2;
}
```

In the short example above a section `Base` is defined containing the keywords `a` and `b`. The section `Extended` now inherites the properties of section `Base` and the keyword `a` is overridden. The keyword `b` within section `Extended` now evaluates to 3.

DATABASE SYSTEM

The Input Deck database (see Fig. 1) comes with a toolkit which consists of a reader and a writer module for reading and writing files.

To extend the functionality of the IPL the application may register an arbitrary number of application specific functions to the function module (see Section). Once the IPL files are read and the database is built up the application can query items (e.g. model parameters, simulation flow descriptions, user flags) in the database to test their existence, evaluate parameters, or analyze structures. Moreover, the application may modify existing items or even add new items to the database to write runtime information back into the database. Equations are evaluated by the runtime evaluation module (see Section).

The Input Deck database is controlled by the application via an application interface which is available for

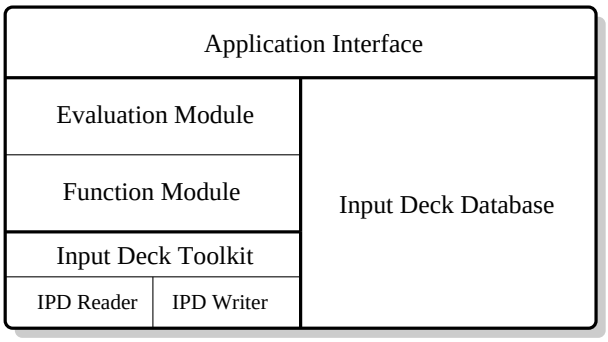

Figure 1: The Input Deck database structure

the programming languages C, C++, and LISP to support several different TCAD applications(Binder et al. 1998) (Binder and Selberherr 2000) (Strasser et al. 1997).

Evaluation Module

Expressions stored in variables may contain references to other variables which allows to describe dependencies between equations. Since the application may modify variables declared extern, so called keywords, at any time, depending variables change too. Thus, variables are evaluated at time of query. A dependence tree is used to check the dependencies. Therefore, circular references are recognized and prevented.

Items stored in the Input Deck database are not assigned a specific type. The type of the value and the value itself are evaluated at runtime by the evaluation module. There are different data types for values: integer numbers, real numbers, complex numbers, real or complex numbers with a unit (quantities), boolean values, and strings. Arrays can be used to store sets of data, e.g., a curve or coordinates of points.

Function Module

The function module manages all functions which are used in IPL. Functions are used within expressions and are, therefore, called at time of evaluation by the evaluation module. The function module distinguishes three different types of functions depending on where they are defined.

Built-in functions are implemented in the function module which are either mathematical or supplementary functions. Mathematical functions are defined for complex numbers in general. Supplementary functions are conversion functions and special functions. Conversion functions are used to convert the result of an expression to a given value type. Some special functions can be used, e.g., to extract the unit of an expressions value, to extract the real or imaginary part of a complex number, or to get the value of an environment variable. The `if()` function can be used for conditional evaluations.

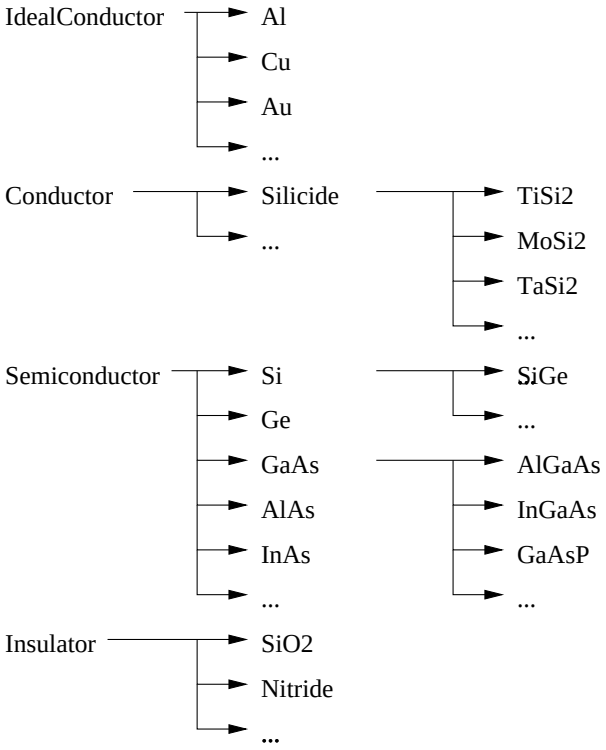

Figure 2: Material Database

Application specific functions are used to extend the functionality of the Input Deck database. They are coded in the application itself. The application can register an arbitrary number of functions. The functions are called by the evaluation module when they are used in expressions. This enables the usage of call-back functions.

User-defined functions can be easily defined in IPL. Thus, the functionality of the programming language can be simply enlarged and customized.

EXAMPLE: MATERIAL DATABASE

MINIMOS-NT offers a variety of physical models which are managed by a separate module of the simulator, the model server(Mlekus and Selberherr 1998). In addition, the model server allows the user to add customized models. Since the models to use depend on the type of the material, efficient material handling becomes an important issue. MINIMOS-NT handles the materials in an abstracted way by using only the class of the material. Supported classes are `Semiconductor`, `Insulator`, `Conductor`, and `IdealConductor`. The properties of the material classes are described via a model set, for instance the mobility or relaxation times of semiconductors. These models are defined as black boxes using a set of input and output parameters. Only the actual implementation calculates the output values. Several implementations are available, each with an unique set of model-parameters.

The material database (MD) manages hierarchically structured materials (see Fig. 2). To administrate common materials properties, real materials like Si or SiO2 are inherited from abstract materials like semiconductor and insulator. Thus, the material properties of a derived material class are extended and specialized.

Each material is represented as a section providing the parameter values for the physical models which are hierarchically structured. For each model class (e.g. the mass density or the band edge energy) and for each material (e.g. Si or SiGe) certain model instances can be chosen with model-specializer keywords.

For each geometric region in a device all physical models to use and their parameters must be specified in a special section. By inheriting the global MD the default settings are loaded which can be locally overridden.

This hierarchical approach has major advantages:

- As MINIMOS-NT only deals with abstract material classes like semiconductor or insulator new materials can easily be added by inheriting from an abstract material class.
- New models can be easily tested and calibrated by overriding the default values.
- By changing or adding new entries in the global MD the material and model defaults can be easily customized.
- Maintenance is easier compared to hard coded default values.

EXAMPLE: MIXED MODE

MINIMOS-NT has been equipped with circuit simulation capabilities(Grasser 1999) to handle distributed devices, for which the semiconductor equations need to be solved together with compact models in one circuit. For the description of circuits, the inheritance feature of the Input Deck is extensively used. Devices are described by sections which are inherited into the circuit. Each device overrides the contact values of the base section to provide the connection to the circuit.

As an example we consider a five-stage current mode logic (CML) (Treadway 1989) ring oscillator. A CML gate is an emitter coupled logic (ECL) gate stripped of the emitter-follower (Embabi 1993) which provided the power gain for driving external circuits. A single gate of this oscillator is

```
Vt   =  25e-3;
k    =  20;
Vs   =  k * Vt;
It   =  2e-3;
Rc   =  Vs/It;
Vee  = -5.2;
Vbe  =  0.889;
Vref = -Vs/2;
Re   =  (Vbe - Vee)/It;

aux useSimpleVersion = false;

Subcircuits {
  CMLBase {
     Vin     = "";
     VOUT    = "";
     Vout    = fullname(Vout);
     intern1 = fullname(intern1);

     Rc1 : ~Devices.R { N1 = "gnd";  N2 = ^VOUT; R = ~Rc;   }
     Rc2 : ~Devices.R { N1 = "gnd";  N2 = ^Vout; R = ~Rc;   }

     T1  : ~MyDevices.NPN { C = ^VOUT; B = ^Vin;   E = ^intern1;  }
     T2  : ~MyDevices.NPN { C = ^Vout; B = "Vref"; E = ^intern1;  }
  }
  <CMLSimple>  : CMLBase { // The ideal current soure version
     It  : ~Devices.I { P  = ^intern1; M  = "Vee";  I0 = -~It;    }
     Rt  : ~Devices.R { N1 = ^intern1; N2 = "Vee";  R  = 10kOhm;  }
  }
  <CMLComplex> : CMLBase { // The transistor current version
     intern2 = fullname(intern2);
     T3  : ~MyDevices.NPN { C = ^intern1; B = ^intern2; E = "Vee";  }
     T4  : ~MyDevices.NPN { C = ^intern2; B = ^intern2; E = "Vee";  }
     RE  : ~Devices.R { N1 = "gnd"; N2 = ^intern2;  R  = ~Re;  }
  }
  CML : CMLSimple  ? (~useSimpleVersion == true),
        CMLComplex ? (~useSimpleVersion != true);
}

Circuit {
  Inv1 : ~Subcircuits.CML { Vin = "pin1"; VOUT = "pin2"; Vout = "hp1"; intern1 = "hi1"; }
  Inv2 : ~Subcircuits.CML { Vin = "pin2"; VOUT = "pin3"; Vout = "hp2"; intern1 = "hi2"; }
  Inv3 : ~Subcircuits.CML { Vin = "pin3"; VOUT = "pin4"; Vout = "hp3"; intern1 = "hi3"; }
  Inv4 : ~Subcircuits.CML { Vin = "pin4"; VOUT = "pin5"; Vout = "hp4"; intern1 = "hi4"; }
  Inv5 : ~Subcircuits.CML { Vin = "pin5"; VOUT = "pin1"; Vout = "hp5"; intern1 = "hi5"; }

  Vee  : ~Devices.V { P = "Vee";  M = "gnd"; V0 = ~Vee;  }
  Vref : ~Devices.V { P = "Vref"; M = "gnd"; V0 = ~Vref; }
}
```

Figure 3: Input file for a Five-Stage CML Ring Oscillator

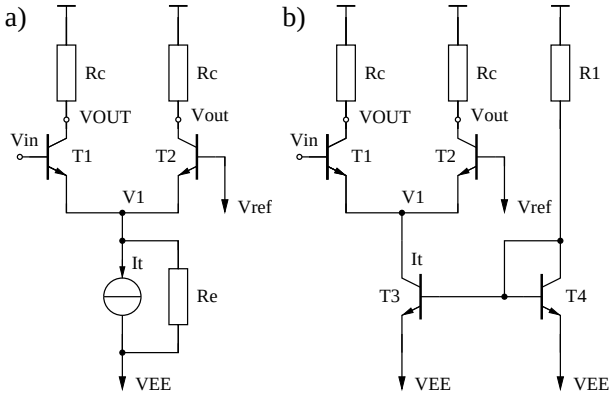

Figure 4: Five-Stage CML Ring Oscillator

shown in Fig. 4. Two different version have been investigated, one with an ideal current source (Fig. 4a) and the other with a more realistic transistor current source (Fig. 4b).

The input deck for the circuit description is shown in Fig. 3. Subcircuits are realized via sections which may contain subcircuits themselves. First, a base circuit is described (`CMLBase`) which contains the differential amplifier. Then the two different current sources are added to this base circuit to give the actual realizations of a CML gate. Via conditional inheritance the subcircuit `CML` is defined using the auxiliary keyword `useSimpleVersion` as a switch. In the actual circuit, the `CML` subcircuit is used five times to form the ring oscillator. Note that via the switch `useSimpleVersion` two different, though related circuits are contained within one input deck which increases the maintainability of large projects.

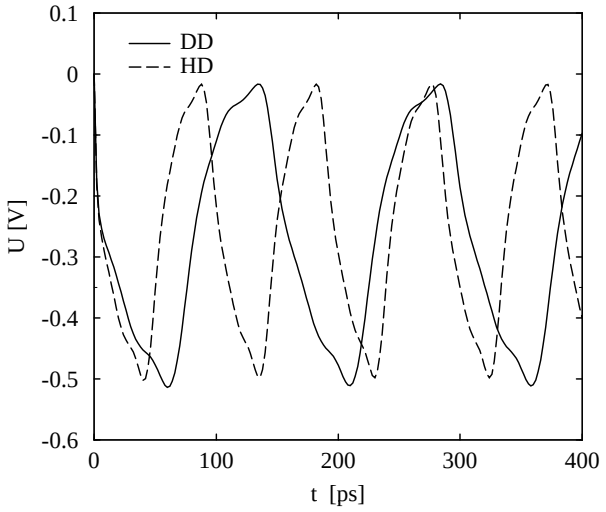

Figure 5: Results

A simulation result is shown in (Fig. 5) where both a drift-diffusion (DD) and a hydrodynamic (HD) transport model have been used. Oscillations start immediately with a frequency $f_{\mathrm{DD}} = 6.8\ GHz$ for DD and $f_{\mathrm{HD}} = 10.6\ GHz$ for HD which gives a relative difference of 36% for the DD model. This is due to the velocity overshoot which occurs in the base-collector space charge region of the transistors which cannot be modeled using a DD transport model.

CONCLUSION

The demands for an input information module for TCAD application rise with the complexity and amount of information. Using a new kind of an object-oriented dynamic database, the Input Deck database, we obtain a maximum of flexibility and maintainability.

The Input Deck has been successfully used to enhance the features of various TCAD tools, especially the circuit/device simulator MINIMOS-NT. With the features of the Input Deck, input decks can be easily customized using auxiliary keywords and environment variables. Furthermore, the material data base has been structured efficiently because of extensive use of the inheritance feature for related materials.

REFERENCES

Binder, T.; K. Dragosits; T. Grasser; R. Klima; M. Knaipp; H. Kosina; R. Mlekus; V. Palankovski; M. Rottinger; G. Schrom; S. Selberherr; and M. Stockinger, 1998, *MINIMOS-NT User's Guide*. Institut für Mikroelektronik.

Binder, T. and S. Selberherr, May 2000, "Object-Oriented Wafer-State Services". In *14th European Simulation Multiconference*, pages 360–364, Ghent, Belgium.

Embabi, S.H.K., 1993, *Digital BiCMOS Integrated Circuit Design*. Kluwer.

Grasser, T., 1999, *Mixed-Mode Device Simulation*. Dissertation, Technische Universität Wien. http://www.iue.tuwien.ac.at/diss/grasser/diss/diss.html.

Klima, R.; T. Grasser; T. Binder; and S. Selberherr, November 2000, "Controlling TCAD Applications with a Dynamic Database". In *Software Engineering and Applications*, pages 103–112, Las Vegas, Nevada, USA.

Mlekus, R. and S. Selberherr, August 1998, "Object-Oriented Algorithm and Model Management". In *Intl. Conf. on Applied Modelling and Simulation*, pages 437–441, Honolulu, Hawaii, USA.

Strasser, R.; Ch. Pichler; and S. Selberherr, October 1997, "VISTA - A Framework for Technology CAD Purposes". In Hahn, W. and A. Lehmann, editors, *9th European Simulation Symposium*, pages 450–454, Passau, Germany. Society for Computer Simulation International.

Treadway, R.L., March 1989, "DC Analysis of Current Mode Logic". *IEEE Circuits and Devices Magazine*, pages 21–35.

AUTHOR BIOGRAPHY

ROBERT KLIMA was born in Vienna, Austria, in 1969. He studied electrical engineering at the Technische Universität Wien, where he received the degree of 'Diplomingenieur' in 1997. He joined the Institute for Microelectronics in September 1997, where he is currently working for his doctoral degree. His scientific interests include device and circuit simulation, computer visualization, and software technology.

TIBOR GRASSER was born in Vienna, Austria, in 1970. He studied communications engineering at the Technische Universität Wien, where he received the 'Diplomingenieur' and the Ph.D. degrees in 1995 and 1999, respectively. He joined the Institute for Microelectronics in April 1996. His current scientific interests include circuit and device simulation, device modeling and physical aspects in general.

SIEGFRIED SELBERHERR was born in Klosterneuburg, Austria, in 1955. He received the degree of 'Diplomingenieur' in electrical engineering and the doctoral degree in technical sciences from the 'Technische Universität Wien' in 1978 and 1981, respectively. Dr. Selberherr has been holding the 'venia docendi' on 'Computer-Aided Design' since 1984. Since 1988 he has been the head of the 'Institut für Mikroelektronik' and since 1999 he has been dean of the 'Fakultät für Elektrotechnik'. His current topics are modeling and simulation of problems for microelectronics engineering.