Foreword

Electronic circuit simulation constantly attracts attention of the scientific community. It is the ever-growing nature of electronic circuits that makes the simulation problem difficult. Algorithms and methods that proves to be efficient for circuits with 100,000 gates, run out of steam when faced with 1 million gates, while others cannot cope with heterogeneous circuits with micro-mechanical and/or optical devices and systems. It is well known that circuit simulation is very resource-intensive and that requirements for simulation of modern electronic circuits are always one step ahead of the state-of-the-art memory and CPU capabilities. Modern ASIC frequently contain analogue, logic, and even microelectromechanical subsystems. In addition, power electronic devices are integrated, driving capacitive as well as inductive loads. Self -heating effects and electrothermal simulation is becoming more and more important. To be able to handle an entire system, a modern simulator must have the ability to express all kinds of sub-systems in the most efficient way, with a desired level of accuracy.

Two main components are to be developed when simulation is considered: the language used for description of the system and simulation requirements, and the simulation engine itself. If we assume that non-electrical devices may be expressed using sets of nonlinear differential equations, the problem of modern simulation essentially requires two different algorithms - one for solving the sets of nonlinear differential equations (ordinary and/or partial) and another for discrete event simulation. It is therefore necessary to have a behavioural simulator which handles mixed-signal systems. Thus, analogue electrical and non-electrical portions of a system are analyzed using accurate and detailed, but quite slow and resource-intensive algorithms, while logic subsystems are expressed using logic values and states, with reverse accuracy and efficiency figures.

There are well established algorithms for simulation in both domains: analogue and digital. Furthermore, there are simulators that are widespread such as PSpice and HILO, to mention two only. What is new in modern developments, is integrated mixed-signal, mixed-level and mixed-domain simulation. Such a simulator is Alecsis 2.3, described in this manual. In addition, a hardware description language AleC++ was developed strongly dedicated to simulation. For that reason, Alecsis 2.3 is an integrated language-simulator system which is able to solve practically all simulation problems arising in electronics and neighbouring areas. Before conclude this foreword the importance of HDLs will be emphasized, so that a short state of the art will be given and AleC++ will be generally outlined.

Hardware description languages using programming language constructs have long been in use in the digital domain. Programming languages have been directly applicable to the task of behavioural description of digital systems as the behaviour of digital systems is naturally expressed in algorithmic form. In the past 30 years, over 200 different languages or environments, each presenting its own conceptual approach to simulating the given problem, have been published. With the introduction of VHDL (Very high speed integrated circuit Hardware Description Language) thing were changed dramatically. VHDL not only became a standard language to describe digital models and behaviour but it is now a basis for development of design tools including synthesis, documentation, graphic representation and so on. Its fundamental philosophy is ADA based. In the same time an additional standard was adopted the so-called Verilog language based on the C language. These two standards provided users with a richer set of tools to produce more reliable models and suppliers with a more rewarding market to explore. Of course, an oft-cited drawback of standards is their tendency to embody the lowest common denominator or to be outmoded by the time they are finalized. But this is certainly not exactly the case when VHDL is considered: it has not only helped to create a competitive digital simulation market but also has forced synthesis and other nonsimulation tools, all with high degree of interoperability. Both VHDL and Verilog now provide a common format for movement toward synthesizers and correct-by-construction tools.

Analogue simulation tools primarily began with early circuit simulation tools, with SPICE being the mainstay of the CAD tools used in analogue IC design process. The newer generation of simulation tools includes modelling at higher levels, and analogue hardware description languages. Modelling techniques such as macromodelling, which earlier were centred around the generation of equivalent circuit models, have now evolved to higher-level analytical models using analytical techniques and symbolic analysis. These higher level models are usually encoded in AHDLs (Analogue Hardware Description Languages). The newer generation of circuit simulators can simulate the mixed-circuit descriptions that combine circuit-level models and higher-level models. Still, among the analogue design bottlenecks one can include:

- the problem of large scale analogue circuit simulation (the performance of digital circuits, having in mind parasitic effects, cannot be evaluated without analogue simulation);
- the simulation of mixed analogue-digital environments which becomes very important in modern telecommunication systems;
- modelling and simulation across various domains (electrical, mechanical, thermal, acoustical, optical, working together in modern robot and medical applications);
- analogue HDL technology (the lack of standard analogue HDL) and so on.

Two early examples of commercial analog HDLs are MAST from Analogy Inc., and HDL-A distributed by Mentor Graphics Corp. A later entry into the proprietary analog HDL marketplace is SpectreHDL from Cadence Design System Inc. The common characteristic of all this languages is relation to a simulator which was developed earlier so becoming some kind of constraint to the language.

The three current standardization efforts that are underway tend to become technology and simulator independent which is similar to the digital counterparts. Chronologically, the first is the Mimic HDL or MHDL sponsored by MDHL Study Group, a part of the IEEE Standards Coordinating Committee – 30 (SSC-30). It was followed by VHDL-A (or VHDL-AMS), developed by the IEEE 1076.1 committee as the analogue extension of VHDL. The most recent is Verilog-A, proposed analogue extension to a co-standard IEEE Verilog language.

New HDL for analogue and mixed-mode descriptions are in development at several universities in parallel. These address specific topics such as verification orientation or component-simulation orientation involving constructs convenient for such system description. Microelectromechanical systems, for example, demand partial differential equation solving, which is not the case in electronic circuit simulation where ordinary nonlinear equations are solved. Having in mind complex interaction which are present in mixed-signal, mixed-level, and mixed-domain designs one intensively considers the object-orientation in both entity overloading and hierarchical modelling point of view.

Object-oriented programming (OOP) is both a general methodology or a way of thinking, and a tool for programming. It is possible to design and write programs based on OOP ideas without any specific OOP language, but a good object-oriented language directs and supports good programming practice.

OOP features can be implemented in any programming language, but some languages are more suitable and flexible than others. The objects serves as an abstraction mechanism where the implementation details are hidden so that the user can rely on a systematic and fairly simple interface between the objects and the external world. The most common features in object-oriented languages are:

- Definition of an object class as a general model or template for the instance objects of the class.
- Hierarchical inheritance of properties (slots and methods) from a superclass to a more specific subclass makes the object-oriented programs systematic and compact by modular construction and code sharing.
- Instances of an object class can be created and deleted at runtime.
- Computation is localized into the objects by defining methods or functions common to a class so that all instances of a class behave in a specific way. These methods functions also form a communication protocol and abstraction mechanism that hides the implementation details.
- A process feature of objects, i.e. defining objects as parallel processes, is useful if concurrent communication between objects is needed.

The most outstanding advantages gained by using object-oriented programming are:

- Natural and clean representation of real world objects.

- Highly modular structure of programs and avoidance of many explicit branching control structures (if then, case, etc.).
- Good reusability of software constructs and modules.
- Easy reconfiguration and maintenance of programs.
- Compactness of programs due to code sharing by inheritance.

Having all this in mind, a language named AleC++ dedicated to simulation purposes was developed as a superset of C++. In addition to the general advantages of OOP mentioned above special constructs were introduced for both control the simulation process, and description of the system to be simulated. One may now use model class inheritance (very successful for hierarchical modelling in IC design environment), entity overloading (used in logic and mixed-signal simulation), clone operation (convenient for repetitive structure description), and many more. All together it may look a bit complicated but it offers much. The reader is advised to look for the introductory booklet "Alecsis the simulator" first. It will give him a glimpse over the whole system enabling better understanding of this Manual.