Published in IEEE Proceedings of the European Design and Test Conference (EDTC) , 535-539, 1994
European Conference on Design Automation (EDAC)
European Test Conference (ETC)
European Event in ASIC Design (EUROASIC)
which should be used for any reference to this work

A Graphical Approach to Analogue Behavioural Modelling

Vincent Moser[1], Pascal Nussbaum[2], Hans Peter Amann[1], Luc Astier[2], Fausto Pellandini[1]

[1]Institute for Microtechnology, University of Neuchâtel
Rue de Tivoli 28, CH-2003 Neuchâtel, Switzerland

[2]Centre Suisse d'Electronique et de Microtechnique SA
Rue de la Maladière 71, CH-2007 Neuchâtel, Switzerland

Abstract

In order to master the growing complexity of analogue electronic systems, modelling and simulation of analogue hardware at various levels is absolutely necessary. This paper presents an original modelling method based on the graphical description of analogue electronic functional blocks. This method is intended to be automated and integrated into a design framework: specialists create behavioural models of existing functional blocks, that can then be used through high-level selection and specification. Applications of behavioural modelling are discussed.

1. Introduction

Behavioural modelling is nowadays widely used for the design of digital systems. The model, typically coded using the IEEE standard language VHDL, is created either by using a high-level specification tool or by manual coding. The design flow can be top-down; after thorough simulation and testing, the models are used for documentation purposes, as specifications for circuit designers and — with some restriction — also as input to automatic synthesis systems [1].

As high-performance analogue and mixed analogue/digital design is very knowledge-intensive, a full automatic top-down approach is very ambitious [2]. Therefore, a meet-in-the-middle strategy can be helpful: specialists create behavioural macro-models of existing functional blocks, accompanied by sets of implementation-dependent parameters, which can then be used by less experienced users through high-level selection and specification tools.

Behavioural modelling has to be introduced into analogue design at various levels of abstraction: at system level and in both top-down or bottom-up approaches.

a) At *system level,* a behavioural model serves as a specification. The model contains an external view of the system (pins, parameters) and describes the desired internal functionality.

b) In a *bottom-up approach,* a model of a newly developed electronic circuit is created, including the external view of the circuit and a functional description. To guarantee behaviour matching between the model and the actual circuit it represents, a set of parameters is carefully defined and the corresponding values are extracted from the circuit through electrical simulation or measurement in laboratory [3]. The model will serve as documentation, hiding the designer's know-how, and, hopefully, allow faster simulation. Behavioural models and the corresponding electrical diagrams then constitute design libraries that are integrated in some surrounding development environment. Models can be used in the design process with the guarantee that a corresponding electrical topology is known: the circuit is realizable in the limits of extracted parameters.

c) In the classical *top-down approach* in system design, specification refinement is done in a hierarchical way [4]. At each level, sub-blocks can be modelled in order to perform system-level simulation. When a sub-block matches an item included in the design libraries, it can be replaced by its schematic counterpart. At that time, behavioural and structural blocks are simulated together; the schema can be sized in order to optimally meet system specifications. Some help should be provided to the user in the selection of the appropriate model according to his specification. An analogue or system synthesis environment could be used to perform this selection task [5].

The scope of this paper is limited to a modelling method for the generation of behavioural models of analogue hardware, which can be applied in any of the three situations listed above. First, the method is globally introduced. It includes a graphical description of the functionality to be modelled, which will hopefully provide independence of the final modelling language (section 3). A correspondence between this graphical representation and available modelling languages is necessary for code generation (section 4). Finally, the results are presented in section 5.

2. Modelling method

Most hardware designers experience coding of analogue models as a difficult and tedious task. Various solutions have been suggested to make analogue modelling available to non-experts and to speed up model generation. Some are graphical but still require good knowledge of the language syntax [6]. Some are fully automatic but limited to particular applications [7]. A new method has been developed that hides code editing to the user and that is widely applicable. A component is successively described in different formats namely a definition view, a functional representation and HDL code. A graphical description kit has been developed that allows to automate model generation using the method presented. Furthermore, this method can be used to develop models of non-electrical systems (sensors, actuators): microsystem integration becomes possible.

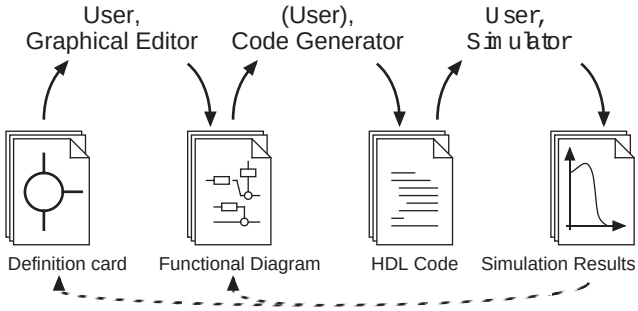

Figure 1: Model development steps

2.1. Global definition

The requirements for a new model are first listed in a textual form: primary characteristics (transfer function, output impedance, etc.) and second order effects (polarization current, PSRR, etc.). According to this specification, an interface is defined in the form of a list of pins and parameters. A graphical symbol, the interface and the list of characteristics constitute the *definition card,* a first view of the model.

2.2. Functionality description

The internal behaviour is then graphically formalized. Symbols, each of which stands for an analytical function, are interconnected using an existing schematic entry tool. This second view of the model, the *functional diagram,* gathers information on the specified behaviour and on the foreseen code structure.

2.3. Code generation and simulation

Each graphical element corresponds to a piece of generic HDL code. The complete *model code* can therefore be derived from the functional diagram by concatenation of the different segments. After compilation, the model is embedded in a test environment and simulations are performed in order to compare the actual behaviour with the specifications given in the definition card. Since no standard AHDL (Analogue Hardware Description Language) is available yet, ANACAD's ELDO-FAS language is used. Nevertheless, the graphical elements defined are general enough — starting from the same functional diagram — for various HDLs to be supported.

2.4. Model check

Using a flexible, automatic characterization tool [8], [9], the validity of the behavioural model generated can be verified. In order to do this, the characterization tool will surround the model with some extraction rigs and perform many analogue simulation runs in order to extract the model instance parameters. If the model runs correctly, the values extracted should match the ones assigned to the input parameters.

This method can also be used to determine the range of validity of models.

3. Graphical description

A formalism has been developed in order to represent graphically the behaviour of various analogue systems. The basic elements will be described first, followed by the connection possibilities and by some typical constructs.

3.1. Graphical Building Symbols

Graphical Building Symbols (GBS) are the primary elements of the formalism, each of which representing an elementary behaviour. They consist of an icon that symbolizes its functionality and of ports used to connect symbols together. Nets are formed, that correspond to signals.

GBS have a set of properties that allows dimensioning of the model and that matches the parameters given in the definition card. Three main categories can be defined: interface elements, functional elements and mathematical elements. They are described in details below. Moreover, GBS can be hierarchical.

a) *Interface elements* allow the model to communicate with the outside world (additional circuitry, user, simulator).

Behavioural models are written in description languages that deal with variables of different types (real, integer, boolean, etc.) but not directly with physical quantities, like most electrical simulators do. Therefore, particular elements are necessary to convert an input quantity into a mathematical entity and, conversely, a calculated result into an output quantity.

• A two port *pin symbol* represents any bi-directional

pin of a system (electrical pin, motor axle) while *conversion symbols* bind that pin to the inside of the model. It is possible to read a quantity on a pin (e.g. current or voltage probe) or to impose a quantity to it (e.g. current or voltage generators). For the extension to non-electrical system, new conversion symbols alone have to be defined (e.g. torque, angular velocity probes and generators).

Besides, parameters might be directly transmitted to the model as pure numbers.

• A *parameter symbol* is introduced representing parameters that are not bound to any particular GBS. From the model point of view, it is considered as an external source of constant numbers. One single port and no conversion element is needed.

• *Simulation variable symbols* make the simulator's internal quantities like time or temperature available to the model. Their use might unfortunately depend on the simulation engine and consequently limit model portability. Again, a useful variable is delivered through a single port.

b) *Function elements* are used to modify a signal's shape or scale. Those two-port-symbols have various properties defined depending on the complexity of the associated operation.

• Linear (pure gain) or non-linear (e.g. limitation) *gain symbols* are defined.

• *Time/frequency symbols* represent time differentiation, time integration, time delay or transfer functions.

c) *Mathematical elements* are used to combine signals.

• *Adders, multipliers* have a variable number of input ports with operators (+ or -, * or /) directly attached to them.

• A *separator* element also exists that splits negative and positive parts of signals: one input, two outputs and no property.

d) *Function generation elements* are sine, cosine, etc. Inputs are signals standing for the operands of the functions (time, frequency, any signal, etc.). If the input must be a parameter, the use of a parameter symbol is required.

3.2. Connection editing

An existing schematic entry tool is used to draw diagrams using the GBS defined before. However, some rules exist for the interconnection of the symbols' ports:

• Internal variables may still carry information about specific physical quantities, it is important, thus, to apply mathematical operators on signals in a meaningful way. Oil and water will not mix.

• Some ports consume signals (input ports of a symbol) while some other deliver signals (output ports). A net must be bound to one and only one output port.

Once a diagram has been edited, a consistency test can be performed.

3.3. Basic functional constructs

Some basic functional groups can be defined. They are common to many models and hence allow easy re-use of code. In the current subsection, constructs used in analogue electronic design are presented. While the first and the second blocks (input and output stages) represent Ohm's law, the third one (power supply) is an expression of Kirchhof's law on currents. The last one (slew-rate) is more analytical, not bound to a classical electrical law.

• An *input stage* contains one pin, interface elements and an input impedance (*Rin*, *Cin*). The voltage is read on the pin, a current is then imposed according to Ohm's law. (Figure 2). Finally, a variable is delivered representing the voltage on the input pin.

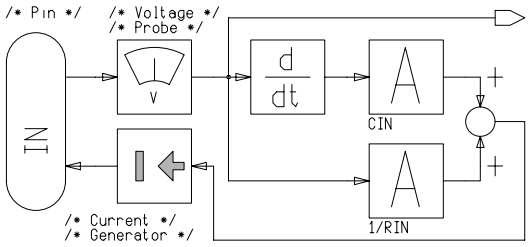

Figure 2: Functional diagram of an input stage

• An *output stage* is composed of one pin, interface elements, an output conductance *Gout* — that may be replaced by an admittance — and an optional current limitation block. The voltage on the pin is read: it represents the voltage after *Gout* while the input variable of the block is the desired voltage. These two values and Ohm's law determine the current that has to be imposed on the pin (figure 3).

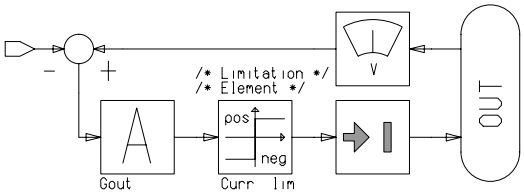

Figure 3: Functional diagram of an output stage

• The *power supply* block includes both usual power supply pins and a polarization pin. The polarization current is computed near to an operating point which depends on the voltage read on the pin (figure 4).

The currents on the other pins are computed by drawing the balance sheet of all the currents in the model:

– All the currents that flow out of the model (except through VSS) originate at VDD.

– All the currents that flow into the model (except through VDD) go to VSS.

An additional loss current is defined as a parameter.

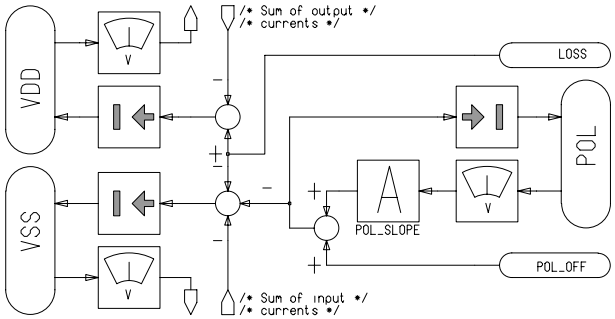

Figure 4: Functional diagram of a power supply block

• The *slew-rate* effect was also modelled. The desired slope of the signal is calculated by dividing the difference between the current value of the signal and its last value by the current time step of the simulation engine. This slope is limited by a maximum rise rate and a maximum fall rate determined by the parameters of the block. The output value is then evaluated according to the computed slope (figure 5).

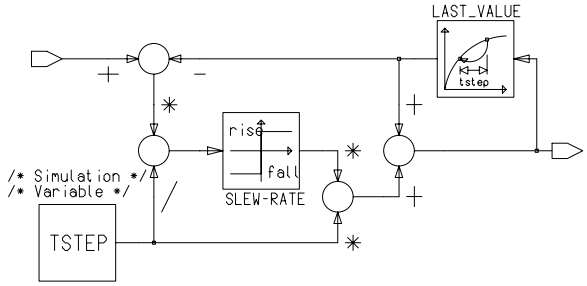

Figure 5: Slew-rate functional diagram

Note: models are simulated using electrical simulators which are time-discrete systems with variable time intervals. A variable delay element (duration: 1 current time step) is introduced in order to get the last computed value of a signal. In the present example, a calculated increase is added to the last value of the output signal.

4. Principles of code generation

In the past section, a general modelling method, including a graphical description set, was discussed. The purpose of this method was to offer comfortable modelling facilities and to ensure model portability and independence of HDLs. The present section deals with the implementation of the model in the form of code. A method is presented for the generation of a complete code file based on a functional diagram and elementary code segments. A detailed example illustrates the approach.

4.1. Method

For the translation of a functional diagram into HDL, a set of elementary generic code segments is necessary, each code segment corresponding to a graphical building symbol (GBS).

The translation process includes the following steps:

• The code segments are collected according to the GBS instances to be found in the design. Property values are introduced.

• Information is organized according to the syntax of the language.

• In order to prevent simulation problems, code segments are ordered with respect to the orientation of the arrows in the functional diagram.

• Connection information extracted from the functional diagram is added in the model code.

4.2. Example

As an example, ELDO-FAS code is generated for the input stage presented in section 3.

A FAS model consist of a title, a declaration part, an initialization part and a body, which can again be split into different parts depending on the analysis mode [10]. The elementary code segments are built the same way except for the title. Generic names are used for the variables.

First, generic code elements corresponding to the symbols present in figure 2 are listed. Declaration part and initialization part are basic and therefore not shown.

• Voltage probe:

```
make v = volt.value(_pin)
```

• Current maker:

```
make curr.on(_pin) = i
```

• Time derivation:

```
if (mode = dc) then
    make yd = 0
else
    make yd = state.dt(y)
endif
```

• Gain

```
make yout = a * yin
```

• Two input adder

```
make yout = y1in + y2in
```

Secondly, the segments are combined and we get the final code:

```
analog
    make v2 = volt.value(in)
    if (mode=dc) then
        make yd4 = 0
    else
        make yd4 = state.dt(v2)
    endif
    make yout5 = cin * yd4
    make yout6 = gin * v2
    make yout7 = yout5 + yout6
    make curr.on(in) = yout7
endanalog
```

This example shows that the generation of a model based on a functional diagram and generic code segments can be done in a very systematic way: the process can be automated.

Note: most electrical simulators are designed to simulate discrete electronic devices. Convergence problems may occur while dealing with behavioural models that usually include discontinuities (e.g. *if…then…else* constructs). Therefore, additional simulation expertise has to be included in the coding process in order to prevent such problems.

5. Results and future work

Several models have already been written in ELDO-FAS using our method. As an example, the model of a triggered comparator is shown. It includes a differential input stage, a fully balanced output stage with current-limitation, a complete power-supply and an extra input for the strobe signal. The slew-rate is also modelled. The corresponding functional diagram is given in figure 6.

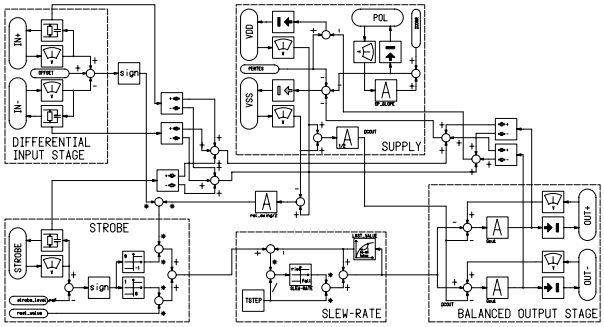

Figure 6: Functional diagram of a comparator

The model has been successfully simulated and the desired behaviour verified. A CMOS comparator described at SPICE level is simulated and the results are compared. Input and output signals are displayed in Figure 7. ELDO needed 4.9s Sun Sparc 10/30 CPU time to simulate the FAS model and 15.2s to simulate the circuit (11 MOS).

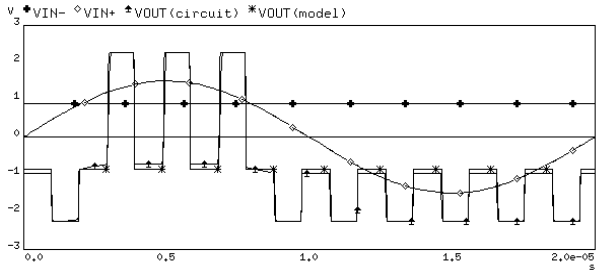

Figure 7: Simulation of a triggered comparator

We now focus our attention on automatic code generation including information extraction from functional diagrams and implementation of the principles presented in section 4. The generation of models in standard VHDL-A or similar language will be of great interest when a compiler and a simulator are available.

A second direction of research is the realization of a set of models that, through their respective parameter sets, match existing blocks of hardware, and the integration of these models into an analogue design framework.

6. Conclusions

The subject of behavioural analogue modelling using a graphical representation has been introduced. A general modelling method has been presented that allows non-experts to develop models. Three representations are used: the definition card, the functional diagram and the final model code. After a definition phase, the functionality is described graphically and then translated into code. A graphical description kit and a method for the generation of HDL code were described in detail.

The positive echo this work received from potential industrial users is promising and encourages us to continue in this challenging research.

7. Acknowledgements

This work has been supported by the Swiss “Fondation pour la Recherche en Microtechnique FSRM” under contract FSRM 91/06.

References

[1] H.P. Amann et al., “The MODES modelling expert system”, Proc. EURO-VHDL'91, pp. 192-195, 1991.

[2] Georges Gielen and Willy Sansen, “Symbolic Analysis for Automated Design of Analog Integrated Circuits”, Kluwer Academic Publishers, 1991.

[3] H. Hamad, “Extraction of Macromodel parameters”, JESSI AC12 Milestone Report, 1993.

[4] J.P. Morin et al. “A Practical Approach to Top/Down Analog Circuit Design”, Proc. ESSCIRC'93, pp. 49-52, 1993.

[5] L. Astier et al., “PlanFrame: A Framework for Design Knowledge Capture”, Proc. EuroAsic'93 User Forum, pp. 97-101, 1993.

[6] ANALOGY Inc., “MAST: Analog Hardware Description Language”, Data sheet MDS-0163 4/93, 1993.

[7] B.A.A. Antao et al., “Automatic Analog Model Generation for Behavioral Simulation”, Proc. IEEE CICC'92, pp. 12.2.1-12.2.4, 1992.

[8] CSEM SA, “SimBoy User's and Reference Manual”, 1993.

[9] S.A. Huss et al., “Automatic Performance Characterization of Analog Functional Blocks”, Analog Integrated Circuits and Signal Processing, vol. 1, nr. 4, pp. 277-286, 1991.

[10] ANACAD EES, “ELDO-FAS Dynamic System Modeling, Version 4.1x”, 1992.