Published in IEEE Proceedings of European Design Automation Conference (EURO-DAC) with EURO-VHDL, 522-527, 1995
which should be used for any reference to this work

Generating VHDL-A-like Models Using ABSynth

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

[1]Institute of 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

A method for the graphical specification and the automatic generation of analogue behavioural models is presented. This method has been implemented as a new software tool called ABSynth. The behaviour of the component to model is described as a functional diagram, which is then automatically translated into a VHDL-A-like analogue hardware description language. No syntax knowledge is necessary and the modelling time is reduced.

1. Introduction

The simulation of analogue electronics is based on models of the components. Historically, only device models were available, which were not supposed to be modified by the user. Analogue hardware description languages now become more and more popular because they allow a designer to write new models and to simulate systems at a higher level of abstraction.

In a top-down design approach, based on a step-by-step system splitting up starting from a high-level specification, the sub-systems can all be described as behavioural models. In a complementary bottom-up approach, parameters are extracted from existing hardware and passed to the models to set up a realistic simulation. Using a mix of behavioural models and SPICE-like structural descriptions of transistor circuits, a multi-level simulation of the whole system is possible.

The standard approach to the development of an analogue model implies:

• source code typing,

• iterative compilation and correction of syntax errors.

However, the syntax of hardware description languages makes their use difficult to non-experts, therefore code generation tools would be welcome, as demonstrated by the success of digital high-level specification tools. To be used efficiently and to benefit from existing design software, such tools have to be integrated in commercial design environments.

This paper presents ABSynth (Analogue Behavioural model Synthesizer), the first tool, that generates models in a VHDL-A-like syntax starting from a graphical description of the behaviour [1]. The goal of ABSynth is to provide a simple way to create behavioural models. With ABSynth, the user has to:

• describe the functionality of the component to model as a functional diagram and its interface as an icon,

• generate the model code invoking ABSynth.

While ABSynth has been basically developed for analogue electronic circuits, it has been updated to mixed-nature (e.g. electrical-mechanical) modelling. The extension towards mixed-mode (analogue-digital) modelling including the links with digital tools is under work.

As the IEEE 1076.1 standard analogue extension to VHDL is not available yet, the models are generated in a similar proprietary language.

To improve the user's comfort, ABSynth has been integrated into a design framework.

The structure of ABSynth is shown in the next section, followed by a brief discussion of hardware description languages. The graphical description method and the model generation process are presented in section 4 and section 5 respectively with some results in section 6. Finally, the work planned is described in section 7 followed by our conclusions.

2. ABSynth structure

The structure of the proposed modelling package is presented in figure 1.

First, a schematic editor is used to draw the functional diagram and the icon of the component. Particular building elements are available in libraries. The know-how of the model designer is stored in the database of the graphical editor and can be reused later.

Then, the information necessary to the model generator is extracted from the graphical environment as an EDIF netlist.

The next step is the generation of the model in an hardware description language. There is a second set of libraries is, that contains code templates, which are assembled during the model synthesis phase in order to build the complete model code.

After compilation, this model is stored in a library. It can then be placed in a circuit description together with standard components in order to perform a simulation.

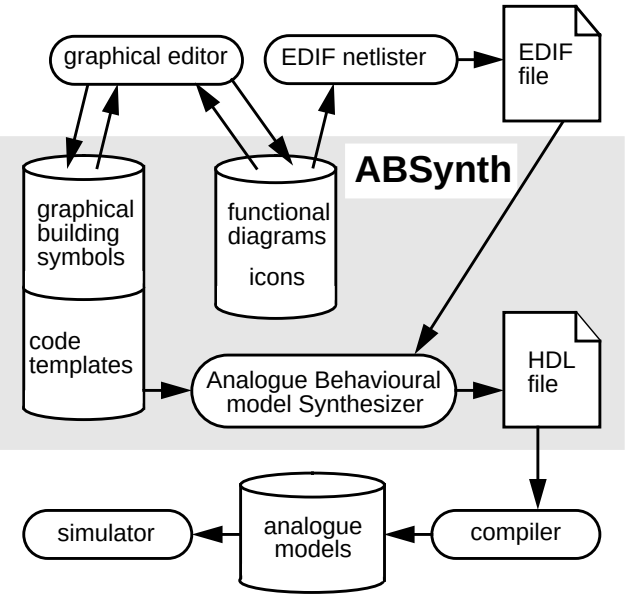

Figure 1. Structure and environment of ABSynth

3. Description languages

Different languages have been used for many years to model the behaviour of dynamic systems [2]. Today, some specific hardware description languages (HDL) are available, that include particular constructs, such as:

• the ability to describe external physical signals as pairs of values; an *across* quantity, which is the representation of a extensive physical quantity, and a *through* quantity, which is the representation of an intensive physical quantity,

• the ability to describe explicit equations as well as implicit equations,

• the ability to describe behaviours with conditional statements.

To ensure the portability of the models, a standard language is necessary. VHDL, an IEEE standard language designed for the description and simulation of digital hardware, is currently being extended to the description of analogue and mixed systems. This future standard is referred to as VHDL-A IEEE 1076.1 [3]. As long as it has not been completely defined and published, we have to work with a commercial product. The current version of ABSynth has been designed to generate models in HDL-A, ANACAD's proprietary language, but we intend to switch to VHDL-A as soon as it is available. The model generator has been written in such a way that it will be easily updated to the new standard. After compilation, the HDL-A models can be integrated in SPICE-like circuit descriptions together with standard components, and the circuits are simulated with ELDO [4].

4. Graphical description method

The graphical description method has already been partly presented in [5]. These features are summarized below while new features are described in more detail.

For the complete graphical specification of an analogue behavioural model, two distinct parts are necessary:

• an icon that gives an external view of the component,

• a functional diagram that describes its internal behaviour.

4.1. The icon of a component

The icon is composed of:

• a body which gives visually a first visual idea of the component's function,

• pins that allow to interconnect this component with other elements in a higher level structural schema; pins can represent analogue electrical quantities as well as quantities of other nature,

• optional properties, that allow a generic model to be set up for a particular application;

4.2. The functional diagram

A Functional Diagram (FD) is an assembly of interconnected Graphical Building Symbols (GBSs). It contains the semantics of the model.

NOTE: The structure of the FD does not match the structure of any particular physical (e.g. electronic) realization of the component. Only the behaviour is described. Furthermore, this structure will not be expressed using structural constructs of the target HDL, but will be translated into a pure behavioural description composed of procedural blocks and equations.

If a component must be modelled differently depending on the analysis type (DC, AC or transient), the behaviour can be described in several analysis-specific functional diagrams.

The functional diagram is drawn with a graphical editor, that allows GBSs to be chosen from a library, placed in

a schema and connected by wires. A standard library of GBSs is available, but the user is free to define additional GBSs. At this level, a wire represents a one-dimensional, non-physical continuous-time state variable. Connection points on GBSs are oriented and thus indicate the direction of propagation of the information throughout the model. The output pin of a GBS can be connected to an unlimited number of input pins but cannot be connected to another output pin (conflict). Some GBSs have properties whose values can be set by the user as expressions that will be evaluated during the initialization of the model. The external parameters of the model (the properties of the component's icon) can be used in such expressions: i.e. a gain GBS that takes 1.0/*ABS_Rout* as the value of the gain property (figure 2).

Once the internal behaviour has been described, pins, that correspond to the ones on the model's icon, can be placed around the diagram. An analogue system is influenced by the quantities on all the surrounding nodes and reciprocally influences them all. For this reason, these pins have to be declared bi-directional.

Particular GBSs compose the interface between the physical quantities on those pins and the internal variables in the behavioural description. Probe GBSs allow the value of a through or across quantity read on a pin to be handed over to variables, whereas generator GBSs allow the value of a variable to be imposed to a pin.

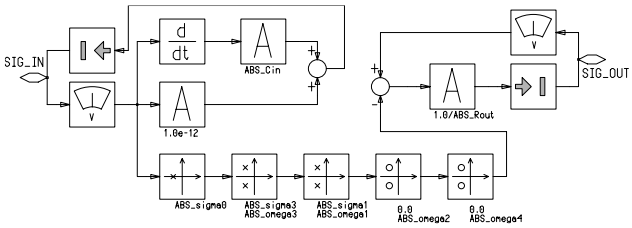

Figure 2. Functional diagram of a 5th order elliptic low-pass filter

As an example, the FD of a 5th order elliptic low-pass filter is given in figure 2. The input stage is characterized by an RC-parallel behaviour, the transfer function is modelled as a series of poles and zeros, the output stage is modelled with an output resistance. The input and output impedances and the position of the poles and zeros are the parameters of the model.

The description of an analogue behaviour by means of a functional diagram has numerous advantages. It is open to people who do not know the syntax of the description language used, it improves the transfer of information between designers and it can be used directly as documentation. Moreover, parts of the functional diagram can be easily reused in later projects.

4.3. Hierarchical design

In order to simplify complex diagrams, the behaviour can be described hierarchically. A part of the functional diagram (a sub-FD) can be replaced by a new graphical building symbol, that represents the same functionality. This new GBS and the corresponding sub-diagram are linked together and placed in the graphical database so that they can be reused in other projects.

In figure 3, the hierarchical GBS of an output stage is shown together with the associated sub-diagram.

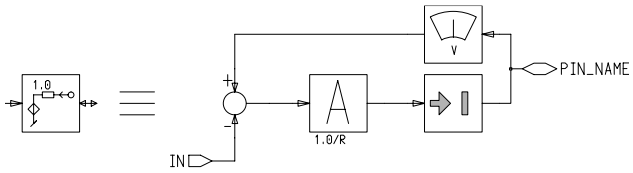

Figure 3. Hierarchical GBS: output stage

The elliptic filter described in the previous section can also be modelled using hierarchical GBSs for the input stage, the output stage and the transfer function (figure 4).

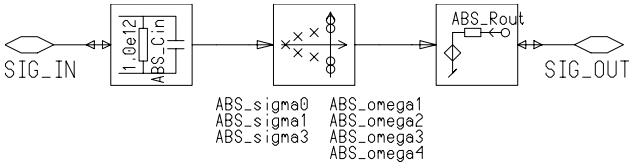

Figure 4. Hierarchical FD of a 5th order elliptic low-pass filter

4.4. New graphical building symbols

The user can extend the GBS library for his own needs. The symbol must be composed of:

- a body which gives a good idea of the corresponding behaviour,
- optional oriented pins,
- optional properties.

The user also has to provide a description of the associated behaviour, either as a sub-FD or as a behavioural code template.

5. Model generation

Two code generators have been developed:

- an entity generator; the entity corresponds to the icon of the component to model,
- an architecture generator; the architecture is the core of the model and corresponds to the functional diagram.

A behavioural architecture will be generated, that cor-

responds to the semantics given by the user in the functional diagram. ABSynth translates this semantics into a code file that follows the target language (i.e. HDL-A) syntax. The necessary syntax information is stored in code templates, each of which holds the same semantics as the corresponding GBS. Alternatively, the semantics of a (hierachical) GBS can be given as a sub-FD.

5.1. Extraction of graphical information

The graphical description must be read and the useful information must be stored in an intermediate format. The EDIF standard format has been chosen in order to remain independent from the graphical entry tool.

The EDIF description of the functional diagram must be hierarchical and include:

• the name of the design, which is the name of the component to model,

• a declaration of each cell used in the design; a cell in EDIF corresponds to a GBS,

• for each hierarchical cell (including the top-level cell representing the actual design), a list of the instances present in the underlying schematic and a list of nets (connection information).

The purely graphical information (symbols, colours, positions) is not useful for the code generation and can therefore be omitted in this EDIF description.

5.2. HDL-A entity generation

The name of the entity generated is the name of the top cell (that is, the name of the whole design). An external connection point becomes an HDL-A PIN; an icon's property becomes an HDL-A GENERIC.

5.3. HDL-A architecture generation

The name of the architecture generated is the name of the view associated to the top cell.

The code generation process is as follows:

First, the semantic information corresponding to the GBSs used is collected. ABSynth looks for a code template in a library structure. If no code template is available, it looks for an associated sub-FD and uses this information to generate a new code template. This new code template can be stored in a library for later reuse.

All the GBS instances are then placed in a table and sorted according to their respective precedence; feedbacks loops are kept unchanged for they will be solved later on by the simulator. For each instance found in the table, a copy of the corresponding code template is done and the identifiers are modified in order to ensure their uniqueness. The content of all those customized code segments is then gathered into a single file. Finally the information that corresponds to the interconnection between the GBSs is added to the code as simple assignment statements; the value of an output variable is assigned to the input variable of the next instance.

The complete model generation, including EDIF netlisting, can be called as a single UNIX command. Various options have been implemented for hierarchy control and library management.

Finally, the model can be compiled with *amc*, ANACAD's analogue model compiler. If the functional diagram and the icon have been edited correctly, which is quite easy, no compilation errors occur. The models are syntactically right by construction.

5.4. Integration into a CAD framework

To make the model generation easier and to provide a complete modelling flow, ABSynth has been integrated into Mentor Graphics' Falcon Framework. The tools used are:

• a multipurpose design entry tool called Design Architect, that includes a hierarchical schematic editor and an HDL-A compiler [6],

• an analogue simulation tool called AccusimII, that can simulate circuits including HDL-A models [7].

A new pulldown menu has been defined, that allows the user to generate and compile the model.

The icon of the model can then be placed in the schema of a circuit for simulation. The external parameters of the model (icon's properties) are set up for the particular application in the schema and the AccusimII simulator is launched.

The user can follow the whole modelling flow from model specification to circuit or system simulation in the same CAD environment.

6. Results

To illustrate our approach, a 5th order continuous-time elliptic filter has been modelled in three different ways:

• manually,

• automatically using ABSynth and starting from the flat functional diagram shown in figure 2,

• automatically with ABSynth starting from the hierarchical functional diagram shown in figure 4.

We compare the CPU time (on Sun Sparc 10) necessary to synthesize the models starting from each type of description (Table 1).

In a first run, the three hierarchical GBSs (input stage,

transfer function and output stage) are considered new. The corresponding code templates are not available. ABSynth generates them (and saves them) based upon the sub-FDs. The generation is somewhat slower than in the case of a flat description. In a second run however, the newly saved code templates are reused and the generation time is reduced by a factor two. The resulting code is the same as by the first run.

Model	flat	hierarchical (1st run)	hierarchical (2nd run)
synthesis	24.7s	29.1s	13.9s

Table 1. Model generation time

Table 2 shows the respective sizes of the code we obtain manually and automatically.

Model	manual	flat	hierarchical
# lines	78	280	299
# states	10	20	30
# variables	8	32	35

Table 2. Size of the different variants

Obviously, the code of the model written by hand is the most compact.

Next, the models are compiled and simulated in AC and in transient mode using ELDO. The simulation of the different models gives identical results, that are presented in figure 5.

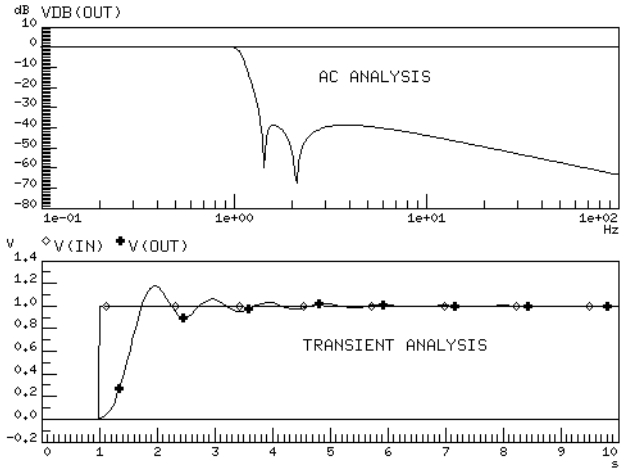

Figure 5. AC and transient simulation of a 5th order elliptic continuous-time filter

The CPU time measured on Sun Sparc 10 in each case is presented in table 3.

Model	manual	flat	hierarchical
compilation	3.9s	6.1s	6.2s
AC	3.6s	6.6s	6.9s
transient	1.5s	2.4s	2.5s

Table 3. Simulation time of the different variants

Both versions of the automatically generated model are simulated in approximately the same time but two times slower than the model written by hand. This difference could be reduced by code optimization, reducing the number of state variables.

The application field of ABSynth is not limited to electrical systems and can also be used to model mixed-nature systems. A piezoresistive pressure sensor has been modelled according to [8]. The characteristic equations of the system are as follows and include offset voltage and temperature dependency:

$$V = (V(Bp)-V(Bn))*(P(inP)*s*(1+tcs*(Temp-tref)) + voff*(1+tco*(Temp-tref)))$$

$$V(outp) = (V(Bp)+V(Bn)+ V)/2$$

$$V(outn) = (V(Bp)+V(Bn)- V)/2$$

The parameters of the model are listed table 4.

Parameter	Definition	Default	Units
s	sensitivity	20	mV/(V*Bar)
tcs	temperature coefficient of the sensitivity	$-2*10^{-3}$	1/˚C
voff	offset voltage	-1.5	mV
tco	temperature coefficient of the offset	$25*10^{-6}$	1/˚C
tref	reference temperature	25	˚C

Table 4. Parameters of the pressure sensor

In the functional diagram given in figure 6, two electrical pins are included for the power supply connections and two other pins for the differential output. Moreover a pin of type *fluid* has been defined for the pressure input. Note the pressure probe GBS, that reads the value of the pressure on *inP*.

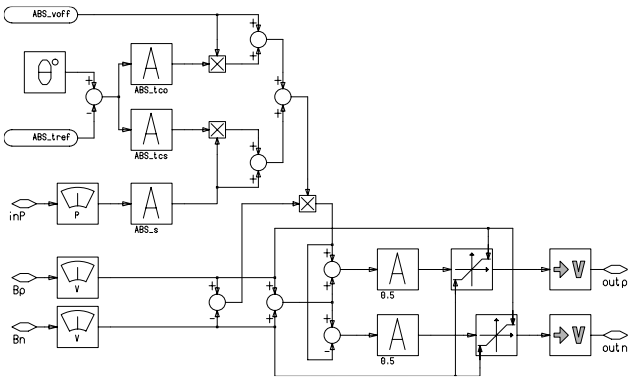

Figure 6. Functional diagram of the pressure sensor

The pressure sensor has been simulated in DC mode with the temperature varying from -50˚C to 100˚C by 1˚C-steps. It took 1.3s CPU on a Sun Sparc 10. Both positive and negative output signals are shown in figure 7.

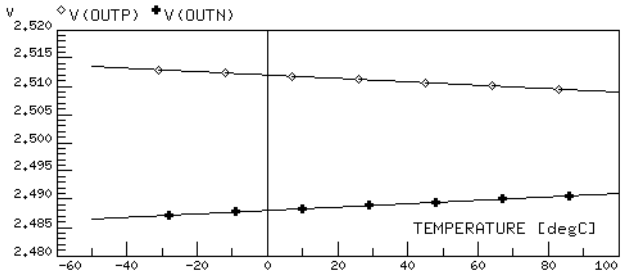

Figure 7. Simulation of the pressure sensor

This example shows that mixed electrical and non-electrical analogue systems can easily be modelled with ABSynth. The expert designer has characterized the sensor using FEM simulation and laboratory measurement. The system engineer, then, builds a simple behavioural model, that can be used with other models for a simulation of a complete microsystem, possibly including analogue, digital, electrical and non-electrical models in the same environment.

7. Limitations and future work

ABSynth is limited to the behavioural modelling of analogue components and systems. Therefore, it is complementary to other tools like Speed's SpeedChart, Comdisco's SPW or Mentor's DSP Station, that generate digital (i.e. traditional VHDL) models.

Improvements of ABSynth are foreseen in particular on the following topics:

• implementation of more language constructs (loops, couplings, direct implicit equation editing),

• implementation of structural descriptions, once the target language supports it,

• mixed-mode modelling including the study of the coupling of ABSynth with digital modelling tools.

Some optimization would also be possible in the form and size of the generated code as well as in the implementation of the code generator itself.

8. Conclusions

A graphical method for the specification of analogue components has been implemented in the form of an analogue behavioural model synthesizer called ABSynth. The specification of the behaviour is done intuitively but still precisely, by drawing a functional diagram. The corresponding HDL-A code is then automatically generated. The tool has been integrated into Mentor's Falcon Framework in order to make the whole modelling flow from model specification to simulation easier. Two application examples have been presented: the model of a continuous-time filter and the model of a mixed-nature pressure sensor.

Acknowledgements

This work has been supported by the "Fondation Suisse pour la Recherche en Microtechnique" under contract FSRM 91/06 and FSRM 91/06P. We thank our colleagues Luc Astier (CSEM) and Ruud Riem-Vis (IMT) for their active help and interesting comments.

References

[1] IMT Uni NE and CSEM SA, *ABSynth User's guide V1.0*, (IMT report 380 PE 01/95), 1995

[2] F. E. Cellier, *Continuous System Modeling*, Springer-Verlag, New-York, 1991

[3] C.-J. R. Shi and A. Vachoux, "VHDL-A Design Objectives and rationale", in *Modeling in Analog Design*, J.-M. Bergé et al., eds, Kluwer, pp. 1-30, 1995

[4] ANACAD EES, *HDL-A User's Manual V1.0*, 1994

[5] V. Moser, P. Nussbaum, H. P. Amann, L. Astier and F. Pellandini, "A Graphical Approach to Analogue Behavioural Modelling", in *Proc. EDAC-ETC-EURO-ASIC 1994*, pp. 535-539, Paris, February/March 1994

[6] Mentor Graphics Inc., *Design Architect User's Manual V8.4*, 1994

[7] Mentor Graphics Inc., *Analog Simulators User's Manual V8.4*, 1994

[8] B. Kloeck, *Design, fabrication and characterization of piezoresistive pressure sensors, including the study of electrochemical etch-stop*, PhD dissertation, University of Neuchâtel, 1989