cuit. This is caused by the semantic nondeterminism of operators or functions. However this problem should be resolvable while providing the context to the synthesis tool.

4) Requirements on Behavioral Description

We expect the language to provide a multi domain and multi level support. This is a requirement arising from the varying descriptions of a behavior as shown in chapter 2.

Most systems, especially those mixing analog and digital signals, only need a time domain simulation facility. This is the most common modelling from.

For classical filter design the frequency domain is needed. Our experience in this field shows us that not only the s-domain (linear) is highly recommended but also the nonlinear frequency domain.

A closer look to digital filtering and SC-technologies shows that a time discrete transformation to the z-domain is required.

From the point of view of electronic engineering a more sophisticated description is required for describing geometry of a component, their port definitions and material properties (dielectric constants for example). These descriptions are needed when EMC simulation have to be carried out.

Last but not least it must be possible to express all properties required for a thermal analysis within the language. For that reason we recommend to add definitions of the relationship between thermal and electrical behavior.

3) Requirements on Documentation

We subsume in this chapter all properties of the language concerning to the user interface of the language. Due to the assumption that the language describes the hardware in a textual way we define this as information layer. A user-friendly information layer will demand more than one description of the hardware rather then a strict description as we show in the chapter above.

The first requirement which has to be stated is the readability of the language. Arising from our experience with EDIF and VHDL we recommend a language design which is oriented to a imperative language design. Nowadays, most developers doing system simulation normally try to formulate their problems in ‘C’ or Fortran. If the language follows the guideline of an imperative language they need not to be trained in an other programming paradigm. This will increase the acceptance of the new language.

A today slogan called object oriented language will now be investigated for suitability to hardware design languages. Due to the efforts to put an object oriented approach to VHDL there must be found some facts.
Two key features are characterizing an object oriented language. An object is build up from data and methods manipulating the data and there is no way (in principle) to manipulate the data except these defined methods. This is called encapsulation. The next key feature is called inheritance. This feature allows to create classes founded on existing ones while ‘inheriting’ all properties and methods of the base class.
These features all address the following problems. The first is the reusability of the code. This problem seems to be solved with inheritance. If we try to translate this approach to hardware description we can state that rewriting a ‘method’ of a class (circuit) means to rewrite the behavior. This is identical to reimplement the whole model because our objects normally consists of only one method. Even if you think that a model can have different methods for different sort of simulations it is also true that every method has to be reimplemented. Therefore, inheritance has no impact on the reusability of the code for hardware modelling. This result doesn’t surprise because hardware description itself is object oriented (there are some instances of components (objects) communicating via messages (signals)!). But there is neither any need to give the language an overhead of a class concept nor do we have to supply constructors or destructors. This is due to the fact that hardware normally behaves very static in sense of lifetime while simulating.
On the other side we found function overloading to be very useful for the readability of the text. But we also have to state that function or operator overloading implies trouble when trying to synthesize a cir-

All these description forms have to be a part of AHDL. Due to the restriction that an AHDL must be a textual description there must be a way to transform all graphical representations towards textual representations. When considering the statement that all systems can be described with the interconnection and the behavioral functionality, we also may state that this transformation is a sort of preprocessing. The only impact on the language itself is to provide appropriate data types. We demand therefore a strict nomenclature and build-in data types for these kind of descriptions.

2) Design representation

This chapter gives an overview of different design representation types. Due to the problem oriented way when describing a system, there are naturally very different approaches.

One of the obvious types of description is a schematic. With the point of view arising from the graph theory you will find nodes representing the function of a component and the edges representing an ideal connection in-between nodes. This abstracted representation form provides some difficulties when the system modelling level begins to mismatch the needs. For example the ideal wire interconnecting components cause modelling problems in high frequency areas. Mostly the whole approach of nondistributed components provides problems in such cases.

For most state oriented representations a signal flow graph is a very convenient way. In this case the nodes of the graph are identified with the interconnection and the edges will represent a transfer function. This kind of description is very useful in the area of data driven systems. But it fails in describing adaptive systems like adaptive filters and on-line changing transfer functions.

In the high frequency area we find system representations via S-parameter. The S-parameter description is able to handle distributed components. It is also possible to model nonideal interconnections. Furthermore systems can be modelled with the help of state diagrams, differential equations (linear and nonlinear), pol-zero diagrams, transfer functions or two port descriptions.

Table 1: Design representation forms

form	kind of representation
schematic	graphical
behavioral	text
signal flow graph	graphical
transfer function	text
two port	mixed text, graphical
S-parameter	mixed text, graphical
differential equation system	text
state diagram	text
pol-zero diagram	graphical
magnitude/phase diagram	graphical

one and mostly called behavioral representation. With respect to the representation of a system there are many different descriptions in this field. Commonly representation is done in differential equations or in partial differential equations, respectively.

For simulation it is obvious that only a behavioral model provides executable codes. This can run as stand-alone code or in conjunction with structural information. It is also evident that pure structural representation can not be simulated. On the other hand it is possible to document a circuitry completely with a structural approach. Therefore a common definition of the elements is needed but this is usually provided. We also see that both behavioral and structural description is fully sufficient to describe a system in general.
Based on this perception we can point out that a hardware description has only to consist of a structural and a behavioral representation. Thus means all following items are not part of AHDL and should be handled in a clear distinct way in AHDL. That is one restriction concerning synthesis.

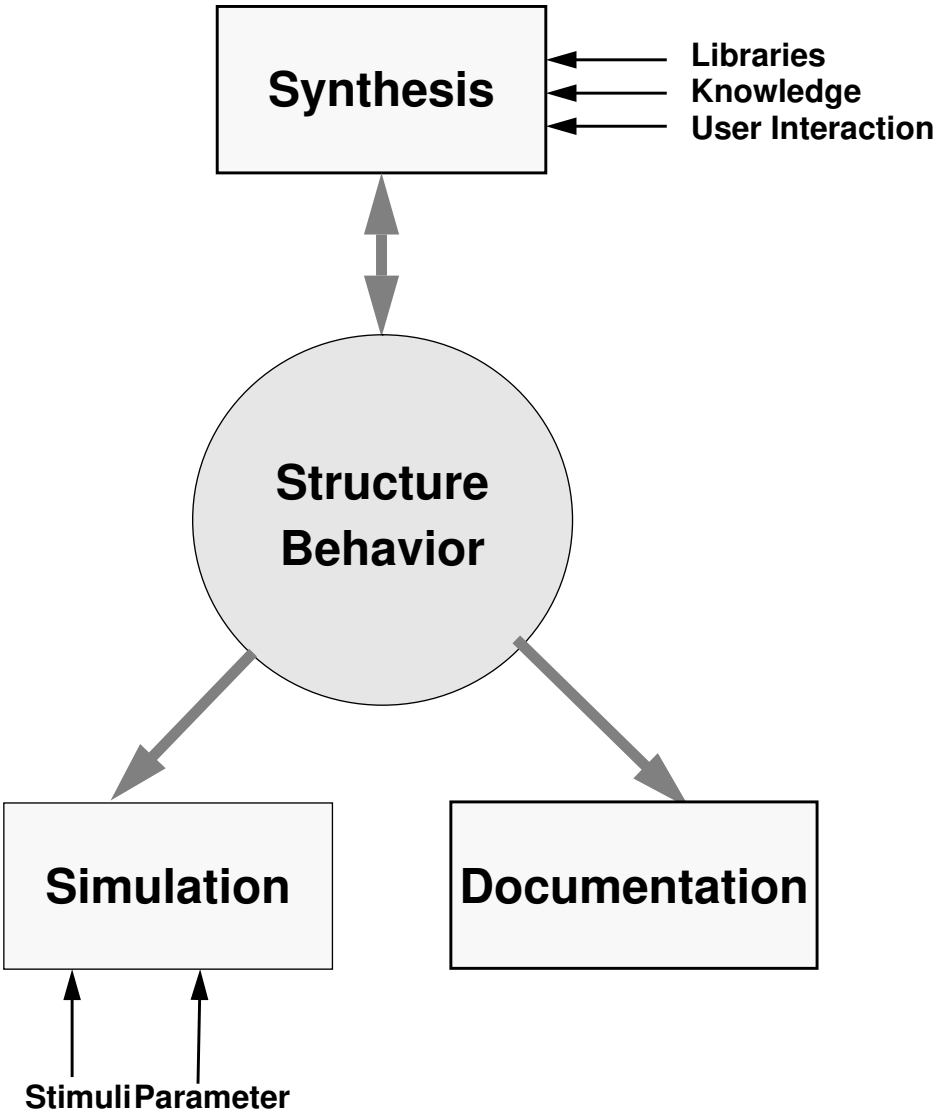
If we simulate a system, the description will be only one part of the required input. We also have to provide stimuli and parameters to the simulator. The way how stimuli or parameters are described or changed for the simulator, addresses the environment of an AHDL. This environment can also be a subject of a standardization activity but it is not substantially for AHDL.

A closer look at synthesis shows us new types of information. In this field the need is emphasized on knowledge representation. Moreover there are some user interactions required. But within an information flow diagram the interaction between tools is also reduced to structural and behavioral description. Therefore the knowledge representation may not be part of the AHDL. Like in the field of simulation this theme impacts the environment of an AHDL and has to be shifted in a standardization activity concerning the environment. Nevertheless leaf cells or libraries should be presented in AHDL to have a common base for simulation.

1) Information flow within AHDL

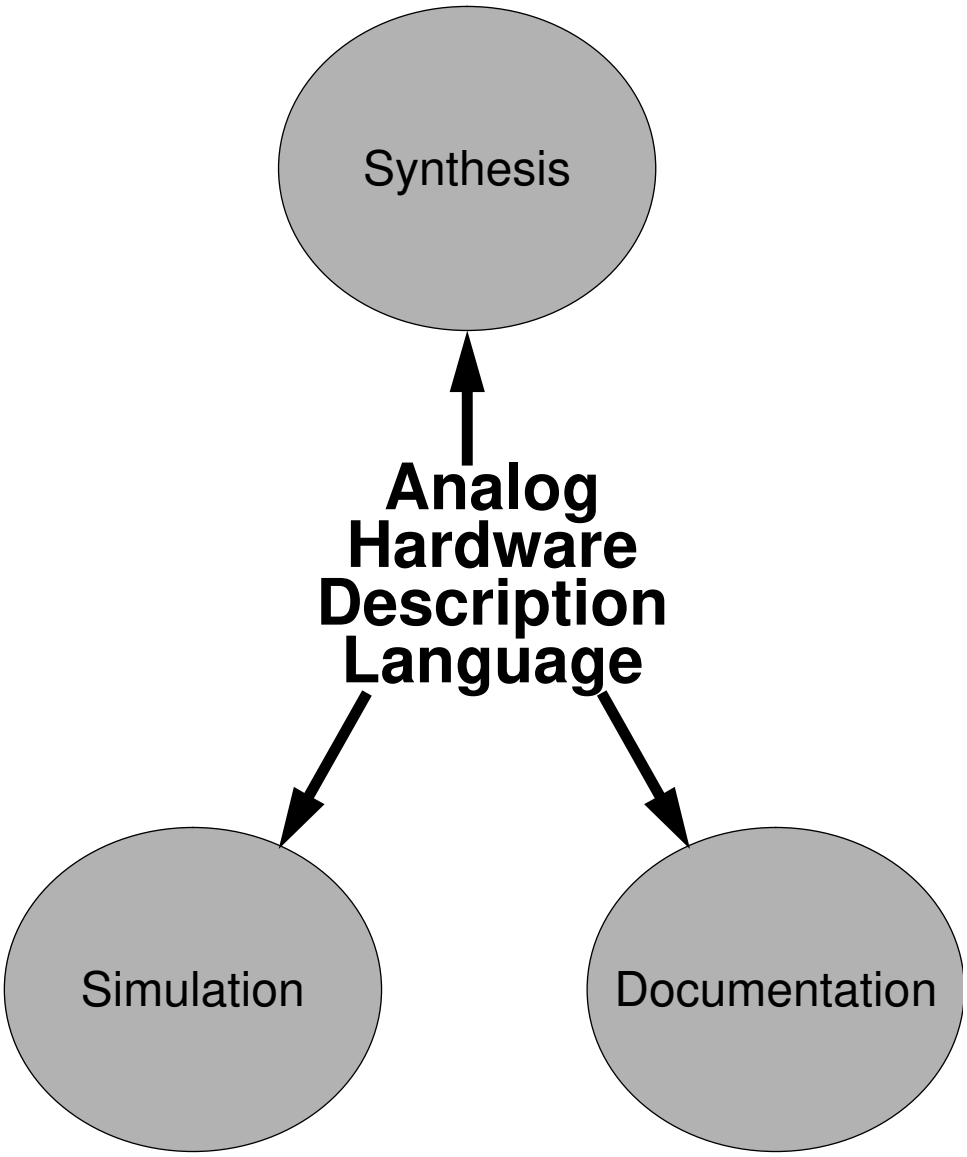
In this chapter you will find some basic requirements, which are the fundamentals for all applications in analog description.

Information flow within a HDL



If we look at every kind of description we find two distinct basic types.
The first type is the structural description of a system.
This type of representation is used for nearly all system descriptions.
In a very natural way the user sketches components and connections, for example in a schematic.
The second type of description is complementary to the structural

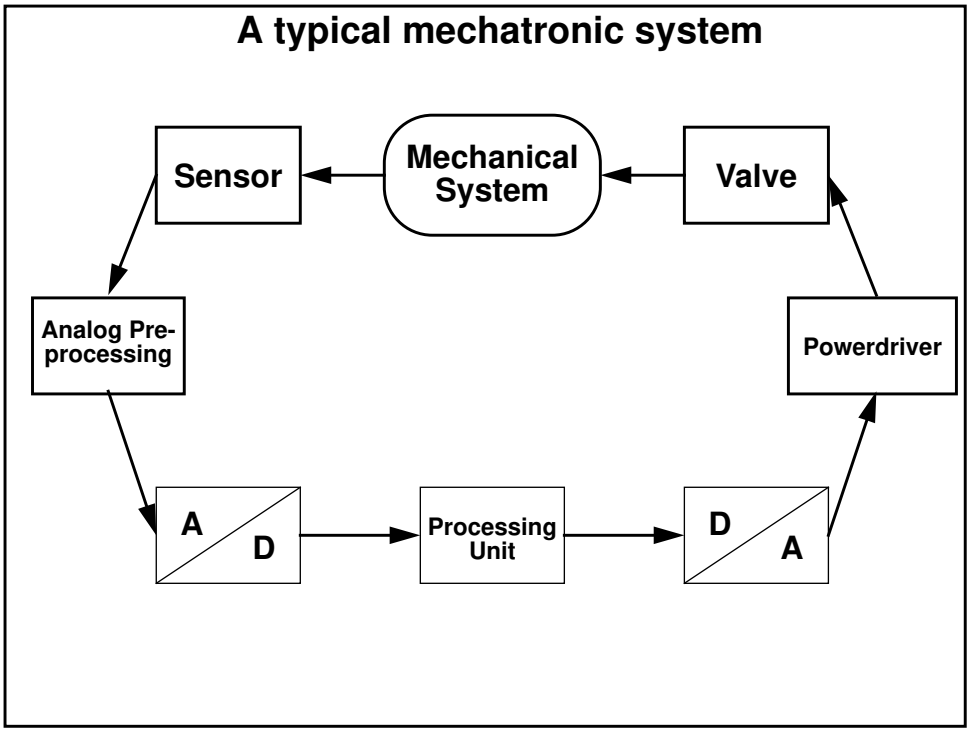
This was found when exploring VHDL, there must also be a efficient way to provide stimuli after partitioning the system. On the other hand we expect AHDL to document in a proper way the system and its realization. A third demand is the support of the requirements due to synthesis in the analog area.



Requirements for AHDL from the point of view of automotive applications

Description of the typical system

A typical automotive control unit contains sensor-elements, some analog to digital converter, a computing unit, a DA-unit and finally a power driver for the actuator. These components are embedded in a control process for engines, mechanical processes or other systems.



Not all control loops are constructed with the complete path via digital processing. There exist also systems which are realized in pure analog signal processing.

On this background we define the requirements for an analog hardware description language. The language must be able to describe all parts shown above for simulation. For the mechanical part this could be a pure behavioral description with differential equations. This concept provides the advantage of describing the system inherent stimuli in the same way as the system itself. The same approach is found within VHDL. This demand does not override the need of having separate stimuli for simulation in the simulator environment.