ATM cell modelling using Objective VHDL*

A. Allara‡
M. Bombana‡
P. Cavalloro‡

‡ Italtel SpA - DRSI/RSC/DMT
20019 Castelletto di Settimo Milanese
Milano, ITALY
Tel: +39 2 43887431
Fax: +39 2 43888593
e-mail: Massimo.Bombana@italtel.it

W. Nebel†
W. Putzke†
M. Radetzki†

†OFFIS Research Institute
Escherweg 2
26121 Oldenburg, GERMANY
Tel: +49 441 798 2988
Fax: +49 441 798 2145
e-mail: radetzki@offis.uni-oldenburg.de

Abstract - High potentialities in terms of abstraction and re-use for hw design are offered by the recently proposed innovative extensions to VHDL, implementing object-oriented techniques. In this paper we evaluate the results of modelling ATM cells in Objective VHDL, exploiting the language features in terms of abstraction and reuse. The selected modules are representatives of highly used components for a wide range of multimedia applications. Entity-architecture classes and abstract data types are considered. Users methodology and benefits are highlighted. The results can be easily extended to other domains where hw design is involved.

I. INTRODUCTION

Microelectronics is a basic technology for the design and the manufacturing of complex telecom systems. This market segment is characterised by a constant search for innovative solutions and by a strong competitive profile. Both these elements impact on the design methodology and the design flow, making it mandatory to minimise the cost and increase the reliability of the final products [1],[2]. From the manufacturer’s point of view these requirements can be reformulated in the following needs:
1. increase the abstraction level of models [3];
2. enhance reusability of models [4],[6].

The former mainly addresses system design. This includes for instance the specification of mixed applications like in cellular telephony, where complex algorithms are computed on standard microprocessors or dedicated architectures. The latter advocates an improved quality of the design process, and addresses more specifically the problems of manufacturers. Both requirements impact strongly on the quality and complexity management of designs.

At manufacturing sites VHDL and Verilog are the most used hw specification languages. Both are able to express design specifications and properties at different levels of abstraction (system, RT level, gate level). Model reusability requires specification techniques to adapt the same model in several designs. VHDL and Verilog support some intrinsic concepts of reusability, but the available mechanisms are not efficient and general enough.

Innovative extensions of VHDL have been recently proposed as an answer to designers’ requirements. Reusability and abstraction in VHDL have been enhanced applying object oriented techniques. The extended specification language, Objective VHDL, described in [5], addresses both RT and system level design. It supports all the classical features of Object-Oriented Design as class concept, inheritance, polymorphism and message passing. Each feature provides sound and unequivocal benefits in term of design style, supporting an extended re-use. The greatest advantage is obtained when different features are combined in the same design. A motivation of object-oriented hardware description can be found in [7].

In this paper we will focus on the results of the application of Objective VHDL in the telecom domain, and specifically to the design of ATM cells. Anyway our results can be easily generalised for other application domains, so addressing a larger volume and revenue than the one considered here.

In section II we will present a quick overview of the features proposed as extensions of VHDL. In section III we will discuss the methodology for the application of entity-architecture classes to the design of a specific ATM cell, highlighting the benefits derived using Objective VHDL in comparison with the traditional approach. In sections IV and V we will analyse how the concept of type class can be applied to model abstract data types and global benefits for design. Finally some conclusions and hints on future work will be presented in section VI.

II. OBJECT ORIENTED EXTENSIONS TO VHDL

A class concept, inheritance of classes, message passing for the communication between classes, and polymorphism

* This research is partially supported by ESPRIT project REQUEST n.20616
are the basic object-oriented concepts. We outline in the following the way they are added to the VHDL data type system and in part to the entity-architecture pair by the Objective VHDL extensions we used for the ATM cell model. The VHDL extensions have been presented in detail in [8] and are formally defined in [5].

A. Data type classes

The concept of a class as a data type is known from the C++ language. Its purpose is to provide encapsulation of data fields (attributes) which belong together logically, like for instance the entries of an ATM cell.

Encapsulation is achieved by limiting access to the attributes to subprograms (methods) of the class, and ensures data consistency provided the methods are correct. Moreover, a class realises the principle of information hiding by separation of the class interface formed by the declarations of methods from the concrete implementation. In order to integrate these principles into VHDL, a class and a class body were added to the available kinds of types (like e.g. record, array, or file).

The class provides primarily the declaration of attributes using the keywords class attribute and methods (i.e. subprograms: procedure, function), whereas the corresponding subprogram bodies are implemented in the class body. Declaration of a class as abstract allows to defer subprogram bodies to derived classes and prevents any instantiation of that class.

A class can be derived from a given parent class (inheritance) with the keywords new class. The derived class can add class attributes as well as subprogram declarations. The class body of the derived class automatically inherits the ancestor’s class body and may add subprogram bodies.

In case an inherited subprogram or class attribute is re-declared by the descendant, it is hidden rather than replaced, and all the code of the parent will continue to work with the hidden version. Thus, its correctness is retained, and the designer of a derived class is responsible solely for the additions made in the derived class.

A class type may be used to declare a signal, variable, or constant. A subprogram of the class can be invoked with such instantiation (object) as a prefix. Thus, the message passing is simply a subprogram call as in C++. The VHDL specifics of signals (e.g. special assignment), constants (no modification), and variables can be accounted for by subprogram declarations or bodies in specific parts of the class (class body, resp.). These parts are declared with the keywords for signal (constant, variable, resp.).

Finally, a signal or variable declared with the type TCLASS, which is implicitly available for each class type T, is the basis of polymorphism. Such signal or variable can contain a value of type T or any class derived from T, and the subprograms of the base class T may be invoked with it.

Depending on the actual class type of the contained value, the inherited or re-declared version of a called subprogram valid for that actual class is chosen for execution. This mechanism (dynamic binding) allows to deal with related classes in a uniform manner, even if they will be derived in a future revision of a system.

B. Entity-architecture classes

Hardware components are traditionally modelled with entity-architecture pairs in VHDL. They have some features not available with the software-like class types presented in the previous section for the purpose of data modelling, in particular the possibility to declare concurrent processes. Thereby, rather than a reactive class type whose methods are being called from outside, an entity-architecture pair can represent an active piece of hardware working in parallel with other components. Hence, to provide object-oriented capabilities also to hardware components, Objective VHDL adds object-oriented features as well to VHDL’s entity and architecture constructs.

The entity-architecture pair can serve as a class since it provides the separation of interface (entity) and implementation (architecture) as well as the protection of its internals against any access from outside the entity-architecture class. Only the keyword abstract, with the same meaning as for the class types, is added by Objective VHDL. Furthermore, through the addition of inheritance by means of the keywords new entity and new architecture, two different kinds of component object-orientation are provided:

- **Structural inheritance** of generics, ports, and concurrent statements. This allows to extend the generic and port interface of the entity, and to model new behaviour in additional concurrent statements. Overriding (as opposed to hiding) of labelled concurrent statements is provided because the modification of a processes behaviour requires to replace it rather than giving it a concurrent companion.
- **High level inheritance** of declarative items, in particular signals, shared variables, and subprogram declarations. For the invocation of entity subprograms, a message passing mechanism has to be modelled by the user.

III. ENTITY-ARCHITECTURE CLASSES

The design style based on object oriented modelling implies an identification of incremental functionalities associated to different versions of a class (device). These functionalities are considered as specialisation of some basic case, which takes the role of super-class, or generic prototype, of an entire family of devices. A specialisation hierarchy is created, proceeding from the most general case to the most specific. In the case of vertical specialisation, only one class is derived from its super-class. A horizontal specialisation is applied when classes, sharing the same super-class (at the same hierarchical level), are differentiated by the alternative presence of functions in order to implement alternative features (different approaches,
algorithms etc.). Vertical and horizontal specialisation are useful when both new and alternative features are required as it will be shown in the following section. The specialisation features are applied to both entities and architectures. In the case of entities this implies adding ports and generic parameters, to handle the additional functionality. In the case of architectures, new processes, internal signals and variables are introduced to handle the new situations.

Abstract classes are useful to define a set of features common to all the derived elements. These classes are never instantiated, i.e. will never provide the definition of a physical device.

A. Modelling example

The previous methodology has been applied to model a cell of the ATM domain [9] that implements a set of interfacing strategies between two ATM cell transfer protocols, the UTOPIA (Universal Test and Operation PHY Interface to ATM) and a proprietary protocol. The module must implement both the receiving and the transmitting functionality. Other functions handled by the module include cell length errors handling and/or parity errors testing in the cell data. Moreover testing functions of the RAM are considered (BIST or SCAN) and both synchronous and asynchronous reset modes of the internal RAM are supported.

In the original design the global functionality was described in two top level architectures composed by more than twenty processes. This specification style makes the process of updating and modifying the cell difficult, so reducing in practice the possible re-use. The new specification in object oriented style includes a hierarchy of eight classes, shown in Figure 1 using the notation of Rumbaugh [10]. Each subclass specialises a feature of its super-class, so providing a more modular design. The classes of the central section of the tree are abstract and represent prototypes (basic functions, and RAM test features) with no receiving or transmitting specialisation. The right part of the tree implements the transmitting features at different levels of functionality (without test, with test, with asynchronous reset). The left part of the tree includes the corresponding receiving functions.

In this modelling style, only one architecture is associated to each entity. Each entity specialises the entity of its super-class, and each architecture is derived from the architecture of its super-class. In this way expressive power is combined with a simple hierarchical framework.

IV. TYPE CLASSES

Abstract data types are widely used in sw domain for modelling complex applications. When behavioural descriptions of components are required, along with structural ones, this technique presents sound advantages also for hw modelling. While entity/architectures classes are clearly applied in structural and behavioural VHDL architectures, data types expand the concept of types and functions defined in VHDL packages. To apply this methodology, the designer must start identifying the type of data peculiar to the chosen application domain under investigation. Then the operations required on such data structures must be formalised and described as functions or procedures. A specialisation hierarchy is introduced also for data types in order to enhance flexibility and accuracy. In fact a structured specification makes easier the updating operations and enhances reusability on different design cases. Package headers contain the declaration of classes and methods, package bodies contain the VHDL code of methods.

A. Modelling example

The ATM cell is modelled as an aggregation of three different parts: a header, a payload (the 48 fields containing the information) and, optionally, a footer. The concept of polymorphism is applied in modelling the ATM cell header sections. The Basic_cell_header class is an abstract class and is specialised to describe the standard header of a generic cell. Subclasses of this class model specialisation, i.e. headers with more features than the standard ones. For instance, some attributes of the Internal_cell_header class represent a complex, proprietary structure, including for instance the Routing, Housekeeping and Sequencing fields. These attributes are conveniently modelled with a new class hierarchy (Figure 2) that has the Basic_field abstract class as root. Finally the footer is modelled as a base class called
Basic_cell_footer from which it is possible to derive further classes as, for instance External_cell_footer.

The classes that implement the header and footer types have a set of methods to set and to read the internal values of the corresponding fields. Methods to perform different types of checking, such as the Header Error Control (HEC) verification, parity test and so on, are also implemented.

The External_cell_header class contains the declaration of a set of methods to read and write all the attributes of the extended application of re-use. Major improvements are expected for cells with at least a possibility of being re-used in three or more designs. Savings in design time can range up to 40% in these cases.

Associated drawbacks are minimal, as only minimal training is needed for the average designer, apart from the knowledge of the syntactic rules for the extended statements and the semantics of the new constructs and the design methodology.

VI. CONCLUSIONS

The design experiences described in this paper show that object oriented VHDL modelling is an effective design methodology to enhance the design standard in terms of industrial and marketing parameters. The Objective VHDL code must then be translated into VHDL code by tools which are still under development at this moment. The efficiency of the obtained translation in terms of overhead in VHDL source code, and more specifically, in terms of area and timing after behavioural or logic synthesis are of paramount importance for the introduction of this methodology. Future trends of research will involve the analysis of the translating tools and the impact on design constraints at low level. At the same time a modelling for abstract specifications involving message passing will be undertaken and advantages evaluated.

REFERENCES


