Exploiting Polymorphism in HW Design: a Case Study in the ATM Domain

Luigi Pomante
CEFRIEL, Via Fucini 2, 20133 Milano (Italy)
pomante@cefriel.it

ABSTRACT

The need of raising the level of abstraction and improving reuse in HW design suggests the adoption of an object-oriented (OO) design methodology based on *SystemC-Plus* (i.e. an enhanced *SystemC*). Such a methodology, developed during the *ODETTE* IST project, allows the exploitation of the key features of the OO paradigm (i.e. information hiding, inheritance, and polymorphism) at the behavioral level of description while guaranteeing synthesizability. In this context, the goal of this paper is to highlight advantages and drawbacks derived from the exploitation of polymorphism in the design of an ATM component: the UTOPIA Cells Handler.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles - *advanced technologies, algorithms implemented in hardware.*

General Terms

Synthesis, Design, Experimentation, Languages, Polymorphism.

Keywords

Polymorphism, ATM, Design, SystemC.

1 INTRODUCTION

Currently, one of the most serious challenges in the telecom market segment is to manage the growing complexity of telecom embedded systems and to reduce the consequent design productivity gap. These elements stimulate a continuous development of enhanced industrial methodologies and design flows exploiting reuse, extended verification, and high-level synthesis. One of the final goals is to raise the abstraction level of the models directly applicable (by means of HDLs) to HW design in order to minimize design time and effort.

SystemC [1] is a system-level design language that allows modeling and simulation of complex HW component. Moreover, according to some *tool-dependent* coding styles (e.g. *Synopsys SystemC Compiler* [4], *Forte Design Systems Cynthesizer* [17]), it is possible to reach the low level steps of an industrial design flow.

A meaningful improvement to such a design modeling strategy is offered by *SystemC-Plus* [2][7][8], that is a SystemC extension combining the benefit of adopting an *object-oriented* (OO) design methodology for systems specification and analysis with the possibility to easily fit *ASIC/FPGA* synthesis flows.

The idea of introducing the OO paradigm in the hardware design practice has been considered several times in the past, mainly by following two dual approaches. The first one is based on extending existing OO programming languages by adding them some capabilities for HW modeling [10][11][12]. The other approach aims to extend existing HDLs with some OO language features [13][14][15].

However, none of these strategies has reached a full success primarily because they focus mainly on simulation and
modeling rather than on synthesis. In other cases, they make use of an OO approach only in order to add HW modeling capabilities to a different language (like for SystemC with respect to C++) but do not allow to a real exploitation of OO features (inheritance, polymorphism, etc.) for modeling and synthesis.

The SystemC-Plus approach aims to get over such limitations in order to allow a real exploitation the OO features in HW design while keeping the high-level synthesis capability. In this context, some previous works [20][21] have coped with the analysis of benefits and drawbacks of following such an approach in the design process. In particular, this work presents an innovative extension of the approach discussed in [21] introducing the exploitation of polymorphism in the design of the *ATM UTOPIA Cells Handler* [3]: benefits and drawbacks of such an approach are discussed in the following.

This paper is structured as follows: Section 2 summarizes the technical features of the proposed design methodology while Section 3 describes the design process applied to the selected ATM component. Next, Section 4 provides some experimental results used to compare the proposed approach with different ones. Finally, Section 5 draws some conclusions, highlighting l benefits and drawbacks of the proposed methodology.

2 OVERVIEW OF SYSTEMC-PLUS

SystemC-Plus is both a *C++* library and a design methodology, based on *SystemC*. It allows the object-oriented description of hardware and mixed systems. Each SystemC-Plus description can be translated into an executable specification for simulation by means of an *ANSI C++* compliant compiler. Additionally, hardware can be directly synthesized from SystemC-Plus specifications, by using a synthesis tool [2] capable to process such descriptions.

Practically, SystemC-Plus (i.e. the language, the methodology and the libraries) enhance SystemC modeling with several features:

- user defined (template) classes;
- inheritance;
- polymorphic objects;
- private interface and communication objects.

Each feature shows benefits [7][8] in term of modeling style, mainly based on extended reuse, while guaranteeing the

synthesizability (a primary manufacturers requirements). This imposes some constraints on the models deriving from the compliance to the language subset supported by the involved synthesis tools. The description of these constraints (*SystemC-Plus Language Subset Specification* [2]) is out of the scope of this paper.
As stated before, the key elements of the *SystemC-Plus Design Methodology* are the libraries and the design flow; for this, both are briefly described in the following.

2.1 SystemC-Plus Libraries

The *SystemC-Plus Libraries* [2] consist of three items. The first is the *OOHW Library* that provides class templates for synthesizable polymorphic and/or guarded objects (i.e. a guarded object is a concurrently accessible objects managed by an automatically-generated arbiter). The second is the *Generic Class Library* that provides basic components for the purpose of reuse within several applications. The last one is the *Application Specific Class Libraries* that consist of a set of *application-specific* classes.
The elements in the *Generic Class Library* are divided into four groups:

- **Data containers**: elements of this group are containers parametric in the data type (e.g. *Memory*, *FIFO*, *Stack*, etc.).
- **Interfaces**: elements of this group provide communication mechanisms between modules and/or processes in the design (e.g. *Peer-to-Peer Channel*).
- **Synchronization classes**: these elements provide basic mechanisms for synchronization and data sharing between modules and/or processes (e.g. *Semaphore*, *Mutex*, *Rendezvous*, etc.).
- **Computational elements**: this group contains objects that are commonly used for complex computation, such as complex numbers data type, vectors, and matrices.

The elements in the *Application-Specific Class Library* have been developed mainly for telecom applications and they are divided into three groups:

- **Extensions of data containers**: elements of this group represent specific data structures that have wide application in the *GSM*, *ATM* and *Internet* domains (*ATM Cell*, *GSM Data Record*, *TCP/IP Datagram/Packet*).
- ***UTOPIA Cells Handler***: the UTOPIA Cells Handler [3] is a module that finds wide application in the *ATM* domain, as receivers, transmitters and data checkers for the *ATM* cells.
- **Aggregates of computations**: this group contains objects that are commonly used in the algorithmic developments of the telecom standards. An example is the coding/decoding *GSM* algorithm that uses the *Viterbi algorithm* for channel equalization in the receiver part of a *GSM Base Station*.

2.2 SystemC-Plus Design Flow

The front-end of the adopted *SystemC-Plus design flow* is a *Synopsys-based* one. Therefore, such a flow includes simulation, synthesis, and co-simulation (when needed). In comparison with the traditional flows, this one has a new layer that sits above the behavioral one (top of Figure 1).
The *SystemC-Plus Compiler* [2] provides the input to the Synopsys synthesis flow used to reach implementation level (right of Figure 1). Such an input can be provided at two different abstraction levels: *RT* (in this case it is given to the *Synopsys Design Compiler*) or behavioral (in this case it is given to *SystemC Compiler*). The abstraction level depends mostly on the adopted style for writing the SystemC-Plus model. The two SystemC-Plus libraries are used as design support in the modeling phase, providing simple pre-defined objects, like data containers, objects for computation, or telecom structures like ATM cells.
Two simulation levels are included in the flow (left of Figure 1).

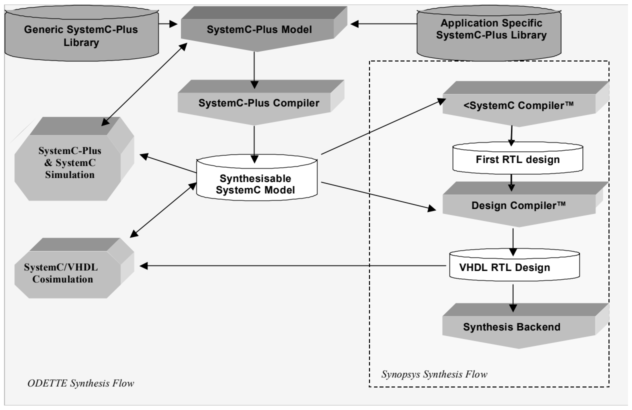

Figure 1. ODETTE synthesis flow

The first simulation is at SystemC-Plus and SystemC level: it verifies the correctness of the functional descriptions provided by the designer. The second simulation is performed at a lower level of the design flow, after the synthesis, to verify the timing constraints and other implementation details. This phase can include a pure *VHDL* simulation, or a SystemC/VHDL co-simulation according to the designer needs. Finally, the design flow reaches the synthesis back-end (based on RT-VHDL), like *FPGA* programming or ASIC net-list generation.

3 A CASE STUDY IN THE ATM DOMAIN

This section describes the OO design process of the *Polymorphic UTOPIA Cells Handler* (P-UTOPIA). First, it discusses the motivations that have lead to the introduction of the polymorphism in such a component, and then it presents the main issues involved in the design process.

3.1 Why Polymorphism?

The *Universal Test & Operations Physical Interface for ATM* (UTOPIA) [3] is a set of modules widely used in the ATM domain to exchange *ATM Cells* among digital systems. The *UTOPIA Cells Handler* defines the interface between the *Physical Layer* (PHY) and the upper ones. The handler performs both the receiving and transmitting functionalities. Moreover, it provides data check functions (e.g. cell length errors, parity error testing, etc.). These components are designed by most of the telecom manufacturers present on the market.
However, the ATM domain focuses on not only elaboration and manipulation of standard ATM cells, but there is also the need to manage custom representations (Figure 2 shows the structure of a real custom ATM cell). For this, most of the HW devices designed to process ATM cells should include complex control sections to manage both the standard cell and one (or more) custom representation. This need arises also for the UTOPIA Cells Handler so each company should design a proprietary version of the handler, according to the constraints deriving from their custom systems. Moreover, greater is the number of different

custom cells that should be managed by a single handler greater is the complexity of the handler itself.

This is a typical scenario where polymorphism could be exploited successfully: the goal is to design a single UTOPIA Cells Handler able to manage concurrently and without modifications any *Custom ATM Cell* derived, in the OO sense, from a basic ATM Cell abstract data type. In other words, the goal is to design a module that relies on methods with standard interfaces but whit an implementation that depends on the effective structure of the custom cell. Such an implementation is so transferred in the design of the custom ATM cell as it is described in the following.

3.2 Virtual ATM Cell

Data aggregates like ATM cells [3] are widely used in many telecom/network applications. These aggregates are used to transfer structured information by means of communication channels. These cells are characterized by a fixed or variable structure made up of standard and/or custom fields (Figure 2), each one carrying a part of the data information to be transferred. These data aggregates have been modeled as *Abstract Data Types* [9][16] providing abstraction and reusability properties by means of methods implementing the operations related to the cell manipulation.

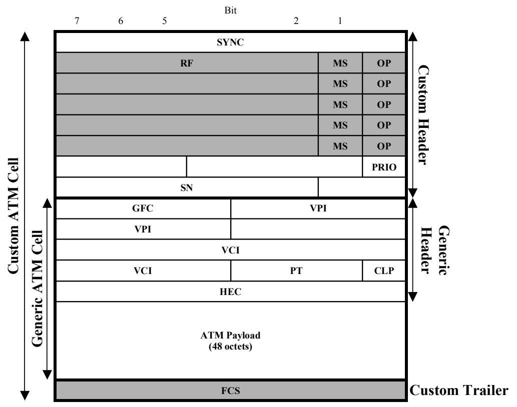

Figure 2. A custom ATM cell

However, in order to be able to exploit polymorphism, the design of the ATM Cell ADT should be modified in order to include the constraint, for each ATM cell, of implementing the methods (i.e. *Split()* and *Assemble()*) used in the P-UTOPIA. Such implementations depend on the effective structure of the cell but have to present a standard interface.

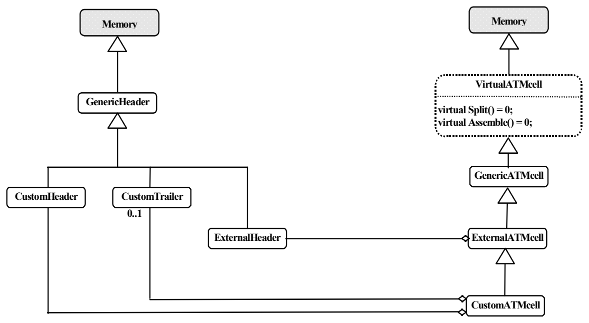

Figure 3. Hierarchical definition of ATM cells

For this, the class diagram in [9] is modified like shown in Figure 3 by introducing in the inheritance relationship a *virtual class* (*VirtualATMCell*) with the *pure virtual* methods *Split()* and *Assemble()*. In this way, every ATM cell must implement its version of such methods. By means of them, it is so possible to model a Polymorphic UTOPIA Cells Handler and, by means of the SystemC-Plus methodology, it is possible to design a synthesizable module (as it is shown in the following).

3.3 Polymorphic UTOPIA Cells Handler

The *UTOPIA Cells Handler* defines the interface between the *Physical Layer* (PHY) and upper layers such as the *ATM Layer*. The handler performs both the receiving and transmitting functionalities. Moreover, it provides some data check functions.

The *SystemC-Plus* model of the *Polymorphic UTOPIA Cells Handler* is composed (as in the non-polymorphic version presented in [21]) of three main parts: an *Input Handler*, a shared data storage unit (this case study considers only the *FIFO-based* version), and an *Output Handler*. The input and output handlers (described in the following) are modeled, exploiting the inheritance feature, as specialized versions of a generic *Event Handler* (Figure 4).

The *shared FIFO* (i.e. a data container belonging to the *Generic Class Library*) is modeled as a *SystemC-Plus Global Object* [2].

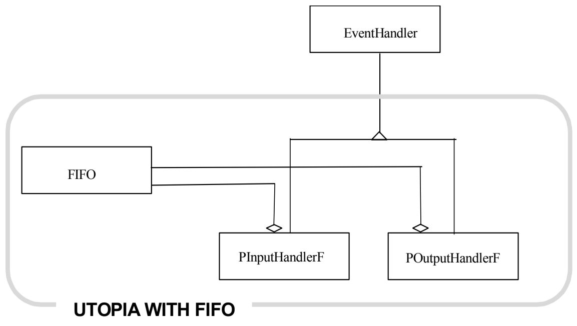

Figure 4. UTOPIA Cells Handler classes diagram

Moreover, the SystemC-Plus template approach (in the C++ sense) allows parameterizing the two handlers in order to increase their reusability. Table 1 shows the template parameters common to the receiver and transmitting parts, while the handlers' design is detailed in the following.

Table 1. Template parameters of the UTOPIA Cells Handler

Parameter	Description	Type
Level	Level selector for data handling (0=Level 1, 1=Level 2)	bool
Nphy	Number of physical layers, up to 32 (n…1)	int
Mode	Supported modes for data I/O (0=8-bit, 1=16-bit)	bool
FSize	Max number of cells in the FIFO	int
SchedulerType	Type of scheduling policy (only for Level 2)	class

3.3.1 I/O Handlers: Modeling and Design Issues

In order to give the flavor about the modeling and design strategies adopted in this case study, the polymorphic input and output handlers are described with more details.

By convention, the signals related to the interface where data flows from PHY to ATM layer are labeled *Rx* (receiver interface, Table 2), while the signals related to the interface where data flows in the opposite direction are labeled *Tx*.

Table 2. Receiver interface of the UTOPIA Cells Handler

Signal name	Description
RxData (15…0)	Input ATM Cell data from PHY to Rx Interface. When 8-bit width interface is selected, data is transferred in the less significant bits (7…0) of the bus.
RxSOC	Input Start Of Cell signal, asserted (active high) when RxData contains the first valid word of the cell.
RxClav	Utopia handshake input control signal. It means Cell Available in a physical layer.
RxEnb*	Utopia data enable signal (active low).
RxAddr (4…0)	Physical layer address. Allows the polling/selection of the physical layer devices connected to the interface (only Level 2). High priority physical layer address is '00000'.
Reset	Input reset.
Clk	Clock for data transfer synchronization, active on the rising edge.

The SystemC-Plus *PInputHandlerF* module implements the receiving functions: by means of the methods described in Table 3, it gets the 8/16-bit width data from the physical layer and assemble the ATM cell (by means of the method provided by the cell itself).

Table 3. PInputHandlerF methods

Member	Description
PInputHandlerF()	Default constructor.
void Polling()	Performs the polling/selection of PHY layers (only Level 2).
void Check()	Enables the module (RxEnb* asserted low) if RxClav is asserted and the module is ready to accept a cell from PHY layer.
void Write()	Reads cells from PHY layer and stores them into the FIFO.
void Assemble()	Calls the proper methods in order to to assemble the cell.

The received ATM cells are stored in the shared FIFO (Figure 5) using the *Write() Guarded Method* [2] that automatically manages concurrent access to the shared data container and checks whether the FIFO is full or not (i.e. from a synthesis point of view this implies an automatic instantiation of an arbiter module).

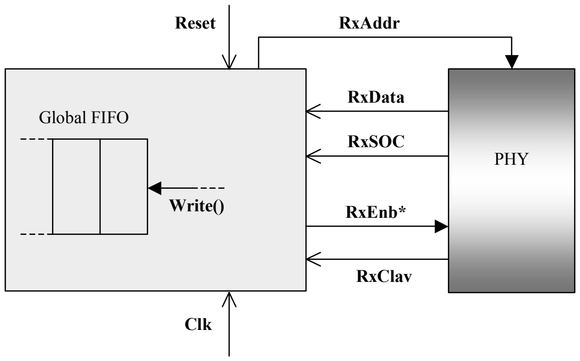

Figure 5. Receiving block architecture

The SystemC-Plus *POutputHandlerF* module (i.e. the transmitter) performs the dual function of the receiver block: it gets ATM Cells from the shared FIFO, splits them into 8/16-bit records (by means of the method provided by the cell itself) and outputs them to the physical layer according to the selected protocol. The cells stored in the shared FIFO are accessed by using the *Get() Guarded Method* [2]. Figure 6 shows a part of a SystemC-Plus header file where it is possible to recognize the input/output interface and the methods of the POutputHandlerF module. It is worth noting that the *VirtualATMCell* class (tagged with the *PolyObject* indication) and the *FSize* parameter are used to instantiate the shared FIFO.

4 EXPERIMENTAL RESULTS

In order to compare the quality of the proposed approach with different ones, the design flow of Figure 1 has been applied to the Polymorphic UTOPIA Cells Handler (using a Sun Microsystems Ultra-5/512 MB RAM and Solaris 2.8). At this purpose, a main program called *Top Module* has been created to instantiate all the necessary library objects and modules. The core of the P-UTOPIA application consists of three modules and 20 classes for a total of about 3000 SystemC-Plus source lines (more than 1500 belongs to the SystemC-Plus libraries).

Next, a testbench has been implemented in order to functionally simulate the specifications before synthesis. Then, the *SystemC-Plus Compiler* [2] has processed the SystemC-Plus description generating the corresponding SystemC description in less than 5 minutes.

```
template
<bool Level, unsigned int Nphy, bool Mode, int Fsize, class SchedulerType>
SC_MODULE(POutputHandlerF)
{
       // External GlobalFIFO
       GlobalObject< SchedulerType, GlobalFIFO< PolyObject<VirtualATMCell>, FSize > > ext_GF;

       // Ports
       sc_out<sc_bv<5> >   TxAddr;       // Physical layer address for polling in Utopia Level 2
       sc_in<sc_logic>     TxClav;
       sc_out<bool>        TxEnb;        // (asserted low)
       sc_out<sc_lv<16> >  TxData;
       sc_out<bool>        TxSOC;
       sc_in<bool>         reset;
       sc_in_clk           clock;

       // … … … Internal variables and signals

       // Methods
       void Polling();     // Performs the polling/selection of PHY layers (only levev 2)
       void Check();       // Enables the module (TxEnb* asserted low) if TxClav is
                           // asserted and the module is ready to send a cell to the PHY layer
       void Read();        // Reads cells from the FIFO
       void Split();       // Splits the cell read from the FIFO in records by invoking the
                           // Split method of the ATM Cell and sends them to the selected PHY layer

       SC_CTOR(POutputHandlerF)
       {
              SC_CTHREAD( Polling, clock.pos() );
              watching( reset.delayed() == true);

              SC_CTHREAD( Check, clock.pos() );
              watching( reset.delayed() == true);

              SC_CTHREAD( Read, clock.pos() );
              watching( reset.delayed() == true);

              SC_CTHREAD( Split, clock.pos() );
              watching( reset.delayed() == true);
       }
};
```

Figure 6. POutputHandlerF SystemC-Plus header file

The output of such a tool is constituted by behavioral and RT level SystemC descriptions. Unfortunately, some imperfections in such descriptions (due to the prototypal nature of the SystemC-Plus Compiler) have lead to a lot of manual work (near 1.5 days) to modify some parts of the code mining the synthesizability of the project.

Hence, the *CoCentric SystemC Compiler* has processed the fixed compilation result. In this phase, the behavioral and the RT files have been compiled producing RT VHDL descriptions. The whole SystemC-Plus to RT VHDL synthesis of the *Polymorphic UTOPIA Cells Handler* has required about 210 minutes. The last phase (*backend*) has concerned with the logic synthesis where the whole set of VHDL files have been synthesized to generate (by means of *Mentor Graphics Leonardo* [6] and *Xilinx Design Manager/EDK* [5]) the bitstream for the target platform: the Xilinx Virtex-II Pro FPGA family [5]. Specifically, it has been used the *Memec Design Virtex-II Pro Development Kit* [19].

Considering a previous work [21], it is possible to compare the presented experimental results with two different non-polymorphic implementations of the UTOPIA Cell Handler: the first is based on *Behavioral SystemC* (according to the Synopsys synthesis guidelines [4]), the second on SystemC-Plus. Table 4 reports a comparison of the main aspects of the three approaches (all based on the same optimizations and a 20 MHz system clock).

Table 4. SystemC vs. SystemC-Plus approach

	Specification Phase (#days)	Developed Code (#code lines)	Behavioral Synthesis (#minutes)	Occupied Area (#LUT)
P-UTOPIA SystemC+	7	1200	210	4300
UTOPIA SystemC+	5	700	160	4050
UTOPIA SystemC	17	3800	45	3100

The main consideration is that the polymorphic SystemC-Plus approach does not present greater costs with respect to the non-polymorphic SystemC-Plus one. The extra days in the specification phase (and the extra developed code size) depend exclusively from the fact that it has been necessary to modify the ATM Cell ADT as described in section 3.2, while in [21] it has been directly taken from the *Application Specific Library*.
Moreover, the increase in synthesis time and area are fully acceptable considering that the module obtained in this way is able to manage concurrently different custom cells. Finally, it is worth noting that to add to the module the capability of manage new custom cells it is needed only the design of the cells with the related *Split()* and *Assemble()* methods (near half a day for each added custom cell). This last consideration highlights the real effectiveness of the polymorphic approach.
With respect to a comparison between SystemC-Plus and SystemC, the same consideration made in previous works [20][21] are true in this one: in the hypothesis of engineered tools and good familiarity of the designer with the methodology, this one could save more than 50% of the design time spent in the specification of a novel system. Even more could be saved in the reuse of existing modules if parameterization and specialization are extensively used. However, there is often a penalty (up to 30%) in area occupation mainly due to the lack of a proper exploitation of the synergies between the different algorithms used by the different synthesis tools. A stronger integration between SystemC-Plus and SystemC compilers could reduce such a penalty.

5 CONCLUSIONS

This paper has presented the OO system-level design features offered by the *SystemC-Plus Methodology* supporting them by a *real-world* case study. In particular, it has focused on polymorphism in HW design analyzing potential benefits and drawbacks. It has been pointed out that the use of polymorphism in the SystemC-Plus design methodology allows the designer to further exploits the reuse of existing projects. Based on the described language, design flow, and libraries, the work presented in this paper will continue experimenting (and evaluating) the use of SystemC-Plus to develop other complex applications.

6 REFERENCES

[1] http://www.systemc.org

[2] http://odette.offis.de

[3] http://www.atmforum.com

[4] http://www.synopsys.com

[5] http://www.xilinx.com

[6] http://www.mentor.com

[7] E. Grimpe, F. Oppenheimer. Object-oriented high level synthesis based on SystemC. *The 8th IEEE International Conference on Electronics, Circuits and Systems*, 2001 (ICECS 2001). Sept. 2001. Volume: 1, 2-5, pages: 529-534.

[8] E. Grimpe, F. Oppenheimer. Extending the SystemC synthesis subset by object-oriented features. *First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. Oct. 1-3, 2003. Pages: 25–30.

[9] A. Allara, M. Bombana, P. Cavalloro, W. Nebel, W. Putzke, M. Radetzki. ATM cell modeling using objective VHDL. *Proceedings of the ASP-DAC Design Automation Conference* (1998). Asia and South Pacific, 10-13 Feb. 1998. Pages: 261-264.

[10] T. Kuhn, W. Rosenstiel. Java based object oriented hardware specification and synthesis. *Proceedings of the ASP-DAC Design Automation Conference* (2000). Asia and South Pacific, 25-28 Jan. 2000. Pages: 579-581.

[11] T. Kuhn, T. Oppold, C. Schulz-Key, M. Winterholer, W. Rosenstiel, M. Edwards, Y. Kashai. Object oriented hardware synthesis and verification. *Proceedings of the 14th International Symposium on System Synthesis*, 30 Sept.-3 Oct. 2001. Pages: 189–194.

[12] S. Vernalde, P. Schaumont, I. Bolsens. Object oriented programming approach for hardware design. *Proceedings of IEEE Computer Society Workshop on VLSI*, 8-9 April 1999. Pages: 68–73.

[13] S. Swamy, A. Molin, B. Covnot. *OO-VHDL: Object-oriented extensions to VHDL*. Computer, Volume: 28, Issue: 10, Oct. 1995, Pages: 18–26.

[14] G. Schumacher, W. Nebel. Object-oriented modeling of parallel hardware systems. *Proceedings of Design, Automation and Test in Europe* (1998). Pages: 234–241.

[15] P.J. Ashenden, P.A. Wilsey, D.E. Martin. *SUAVE: extending VHDL to improve data modeling support*. Design & Test of Computers, IEEE, Volume: 15, Issue: 2, April-June 1998, Pages: 34–44.

[16] L. Pomante, W. Fornaciari, M. Bombana. SystemC-Plus Complex Data Type for Telecom Applications. *Proceedings of Forum on Specification and Design Languages* (FDL 2002), vol. 2, September 2002.

[17] www.forteds.com

[18] L. Cai, D. Gajski. Transaction level modeling: an overview. *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/software Codesign & System Synthesis* (2003). Newport Beach, CA, USA. Pages: 19-24.

[19] www.memecdesign.com

[20] N. Bannow, K. Haug. Evaluation of an object-oriented hardware design methodology for automotive applications. *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*. Volume: 3, Feb. 16-20, 2004. Pages: 268-273.

[21] R. Farina, L. Pizzamiglio, L. Pomante. "System Design using SystemC-Plus: A Case Study in the Telecom Domain", to appear in *International Conference on Cybernetics and Information Technologies, Systems and Applications* (CITSA 2004) and the 10th International Conference on Information Systems Analysis and Synthesis (ISAS 2004), Orlando (USA), July 21-25, 2004.