OBJECT-ORIENTED HIGH LEVEL SYNTHESIS BASED ON SYSTEMC

Eike Grimpe[1], Frank Oppenheimer[2]

[1]OFFIS Research Institute,
Escherweg 2, 26121 Oldenburg - Germany
E-mail: Grimpe@offis.de
[2]University of Oldenburg,
Escherweg 2, 26121 Oldenburg - Germany
E-mail: Frank.Oppenheimer@Informatik.Uni-Oldenburg.de

ABSTRACT: The introduction of object-oriented modelling techniques into the development of hardware seems to open a promising way for mastering the increasing complexity of today's hardware systems. Furthermore it provides the possibility of transferring well known and approved object-oriented modelling techniques from the software development to the hardware development. Since there is a major difference between the nature of software and the nature of hardware the direct adaptation of common object-oriented programming languages to describing hardware is not possible in general. SystemC is a C++ class library and a methodology that introduces some of the missing typical hardware features in C/C++. This paper describes an extended SystemC based methodology and synthesis techniques allowing to use object-oriented concepts like classes, polymorphism and inheritance for the description of synthesisable hardware models.

1. INTORDUCTION

During the past years there have already been several approaches to apply object-oriented modelling techniques to hardware development. These approaches can be divided into two major categories: the first category includes approaches which try to augment existing object-oriented programming languages such as Java [1][2] or C++ [3][4][5] by missing hardware features such as concurrency, reactivity, and an appropriate timing model. The approaches in the second category augment existing hardware description languages like VHDL [6][7] or Verilog [8] by object-oriented features. But none of these approaches includes a seamless, automated synthesis path from an object-oriented high level system description down to a gate level netlist representation, for a broad domain of target applications.

This paper presents a new approach for object-oriented hardware modelling and automatic synthesis of object-oriented hardware models based on an extension of SystemC, called SystemC-Plus. The basic synthesis techniques have been originally developed and demonstrated for Objective VHDL [6] - an object-oriented extension of VHDL - and are now further developed and adapted to SystemC-Plus.

2. OBJECT-ORIENTED HARDWARE MODELLING

The four basic characteristics of object-orientation to be provided for modelling hardware are:

- **Data encapsulation:**
  Data (attributes) belonging semantically together is combined and encapsulated with the primitive operations (methods) processing the data. Attributes together with the methods operating on them form a data type (class). Data access is only possible by means of the appropriate methods, that build the interface of a data type.
- **Inheritance:**
  Inheritance means to create new data types by deriving them from existing ones. Derived data types inherit all the attributes and methods of the types from which they are derived. It is therefore not necessary to re-implement the inherited attributes and methods in a derived type, but they may be refined.
- **Polymorphism:**
  Polymorphism describes a mechanism that allows objects (instances of classes) to change their type dynamically, during runtime of a sys-

0-7803-7057-0/01/$10.00 ©2001 IEEE.

tem. In the majority of cases the possibility to change the type is limited to a certain set of types which are related, e.g. by inheritance, so that some basic properties of an object can always be guaranteed.

- **Communication by method calls:** Communication within an object-oriented system is realised by method calls. A data type is accessed by calling the methods it provides for this purpose, and objects communicate by calling methods on each other.

Though these concepts originate from the software domain, they seem to provide some promising enhancements for the development of hardware, too. By using object-oriented techniques to model hardware designers could increase flexibility and reusability of the modelled systems and their components. Development times and costs could significantly be reduced and the mastery of more complex systems becomes possible. The question still remaining is how to synthesize object-oriented constructs.

An answer to this question is given in [9], based on Objective VHDL. The approach presented there demonstrates, how an object, i.e. its attributes, methods and communication interface, can be represented by a synthesisable model, that additionally allows the use of polymorphism. The practicability of this concept has been proven in the REQUEST project [10] and by means of the development of some prototypic tools working on Objective VHDL. The concepts presented in that approach are general enough to be also applied to other object-oriented languages.

3. SYSTEMC

SystemC [11][12] is a C++ class library and a methodology which allows to model systems, including both hardware and software parts, based on C/C++. For simulating these systems it provides an integrated simulation kernel which is automatically linked to each executable SystemC model. A SystemC hardware model, at least a synthesisable one, looks very similar to an equivalent VHDL or Verilog model. Classes can be found representing entities, ports or signals and special notations allow the modelling of processes like in common HDLs. Furthermore SystemC provides predefined data types for bit, std_logic, bit-vector std_logic-vector and arbitrary sized integers, which typically build the basis of the type system of a hardware description language. The interface of a module is modelled similar to a port list of an VHDL entity and the communication structure between different modules and processes is established by using the signal and port types provided by SystemC.

The appropriate timing behaviour of an executable SystemC model is introduced by the SystemC simulation kernel. This kernel activates processes in accordance with their sensitivity list, it executes statements within different processes concurrently, it schedules assignments to signals and propagates their changed values through the whole system hierarchy. Finally the simulation time is updated. Asynchronous activities can also be modelled, but due to the cycle based execution semantics of the SystemC simulation kernel value changes caused by asynchronous processes become visible not before the beginning of each simulation cycle, which is usually initiated by a rising or falling edge of a global clock.

Actual developments around SystemC indicate, that the used model of time will be significantly revised and refined in future versions as described in the SystemC 2.0 specification [13].

SystemC including all source codes is available under a very liberal Open Source License Agreement based on OSI-Certified IBM Public License and can for example be downloaded from [11]. Under the leadership of important EDA companies the Open SystemC Initiative was founded, with the goal to promote the further development and standardisation of SystemC. Because of its free availability and the possibility to model hardware in C/C++ there is increasing interest in SystemC from the industrial as well as from the academic area.

3.1 Synthesis

It seems obvious that SystemC was originally developed for creating executable C/C++ system specifications early in the design process and not for synthesis. Therefore no official or standardised synthesis subset exists up to now in contrast to VHDL for example. But because of the similarity between a VHDL/Verilog description and its equivalent SystemC description on RT level a manual transformation into one of these languages is fairly simple, as long as a certain coding style which lacks most object-oriented constructs from C++, is adhered to. Nevertheless, several EDA companies like Synopsys [14] and C Level Design

[15] have announced or do already offer synthesis tools for SystemC. Although the concrete synthesis subset of the different tools is not known to the authors the sparely information available suggests, that only SystemC models which make use of a limited subset of C (e.g. excluding pointer, pointer-arithmetic and object-oriented constructs from C++), and which are already looking very similar to an equivalent HDL description are synthesisable with these tools.

4. ODETTE

In summer 2000 the ODETTE project (Object-oriented co-DEsign and functional Test-TEchniques) [16] was started under leadership of the research and development institute OFFIS (Oldenburg, Germany) in close co-operation with the industrial partners Siemens I.C.N. (Milan, Italy), IBM Research Lab (Haifa, Israel), Synopsys Leda (Grenoble, France) and the European Electronic Chips & Systems design initiative ECSI. Goal of this project within the framework of the European commission's IST-program is the development of an object-oriented design methodology for hardware development based on SystemC and the implementation of a synthesis tool for processing object-oriented SystemC models.

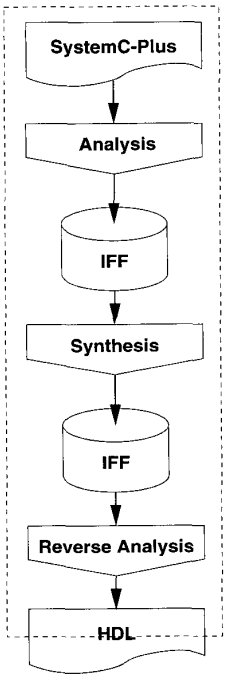

Fig.1. ODETTE synthesis path.

The synthesis tool under development will not directly generate a gate level net list from an object-oriented hardware model written in SystemC-Plus (see next section). Instead it focuses on the synthesis of object-oriented constructs and generates a synthesisable HDL description on RT or behavioural level which can then be further processed with existing logic synthesis tools.

Figure 1 illustrates the synthesis path handled in ODETTE (dashed box). It starts with a SystemC-Plus description, covers several transformations into different intermediate formats (IFF, Intermediate File Format) and ends up with the generation of a synthesisable HDL description, for example written in VHDL or Verilog, which is behavioural equivalent to the SystemC-Plus input description.

4.1 SystemC-Plus

SystemC-Plus was developed as part of the ODETTE project as an extension to SystemC and consists of an additional class library and a coding style which specifies how to write synthesisable object-oriented hardware models with SystemC-Plus. In particular this coding style defines the synthesisable subset of SystemC-Plus, which can be processed by the ODETTE synthesis tool. The additional class library is necessary for using synthesisable polymorphic and so called global objects (see below) in SystemC-Plus.

Though C++ provides polymorphism as a feature, the "native C++" polymorphism mechanism only works by means of pointers, which, besides some special exceptions, are in general not suitable for being synthesized. In addition, the C++ polymorphism mechanism cannot be mapped onto the synthesisable target model presented in [9], which was explicitly developed regarding the implementation of polymorphic objects in hardware. For this reason SystemC-Plus provides class templates and modelling guidelines, which allow to create and use polymorphic instances of any user defined class in a synthesis conformable way. Moreover these polymorphic objects show in a SystemC simulation the same cycle accurate behaviour as their synthesized counterparts.

Therefore SystemC-Plus does not only allow the use of "normal" (non-polymorphic) objects, which are not even supported by most other SystemC based synthesis tools, but also

- **polymorphic local objects;**
  i.e. polymorphic objects declared within any SystemC process,
- **polymorphic ports and signals;**
  i.e. instances of the SystemC type sc_in, sc_out, sc_inout and sc_signal, which enclose a polymorphic class type,

and

- **polymorphic global objects;**
  i.e. polymorphic objects declared as data member of a SystemC module.

Polymorphic signals and ports are quite useful for the abstract modelling of communication channels, which shall be used for exchanging different but (in terms of inheritance) related data.

SystemC, in its actual version, does not provide an appropriate mechanism for handling concurrent accesses by different processes to objects, which are declared as direct data members of a module, and does not seem to be originally designed for such purposes. Method calls from different processes to one common object, which occur in the same simulation cycle, and concurrent read and write accesses are simply executed in a non-deterministic order in the same cycle, but instantly without observable time delay between the different method executions and without causing observable conflicts. This behaviour obviously does not reflect the behaviour of real hardware. Only for some special SystemC signal and port types a rudimentary mechanism for resolving concurrent write accesses is provided, that resembles the so-called "resolved functions" known from VHDL.

For this reason SystemC-Plus introduces the concept of global objects. Global objects are data members of a SystemC module, too, but possess built-in scheduling and arbitration facilities in contrast to normal objects. Global objects are declared by means of elements from the SystemC-Plus class library. In particular a class template for instantiating global objects is provided which requires an arbiter type and any user defined class type to be passed as actual parameters. The passed arbiter type determines which kind of arbiter will be automatically instantiated together with the global object. This arbiter guarantees, that the global object is accessed mutual exclusive; only one client process may access the object at a time, requests of other clients are blocked meanwhile. The temporal behaviour of the hardware to which a global object is mapped and the synthesized communication protocol (see following section) is now correctly reflected in a SystemC simulation, too. In which way concurrent accesses to a global object are scheduled depends on the concrete type of the arbiter that was passed as parameter. Actually the SystemC-Plus class library provides arbiters implementing three different scheduling strategies; round robin, modified round robin and static priority.

It was recognised, that it is desirable to make the permission to access a global object not only dependent on whether other clients also want to access the object at the same time, but to make it additionally dependent on the internal state of the global object itself. If, for instance, a "put" method is called on a global buffer object, which is already completely filled, the calling client should be blocked until new space in the buffer is freed. Likewise, a "get" method call should be blocked if the buffer is empty. For this purpose SystemC-Plus provides a guard mechanism for global objects. Each public method of a user defined class, that shall be instantiated as a global object, must be declared as a so-called "guarded method", following a special notation that is described in the SystemC-Plus coding style. This notation associates a guard condition - a boolean expression - with each guarded method. Since then, execution of the guarded method will be only possible if the corresponding guard condition is evaluated true at calling time, otherwise all calls to the method are blocked until the condition becomes true. This mechanism provides an easy way to realise synchronisation between different client processes, which exchange data via a global object.

Therefore the global objects of SystemC-Plus allow to model communication between different processes, no matter whether located in the same or different modules, on a much higher level of abstraction, than the signal based communication in pure SystemC, without losing the possibility of automatic synthesis.

4.2 Synthesis concepts

The concepts for synthesising object-oriented constructs presented in this paper basically follow [9]. The main idea of this approach is the implementation of objects as finite state machines (FSMs), which can be easily processed by existing synthesis tools. Figure 2 shows a schematic diagram of the circuit being synthesized from a global object

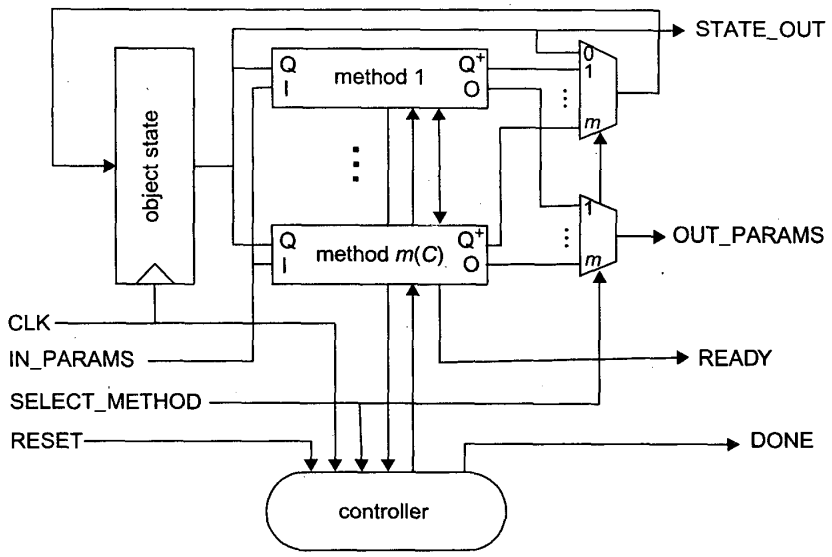

Fig.2. Synthesized target circuit.

(the arbiter is not shown). The data members (attributes) of a class are encoded as bit vectors. The concatenation of all these attributes gives the state vector of an object, which is represented as one single bit vector (object state in the figure). The methods of an object are implemented as combinational networks operating on the object's state vector. Multi-cycle operations are possible, too, by means of a controller, which is also synthesized if necessary. Invoking a method of a global object is only possible through its standardised interface (READY, DONE, IN_PARAMS, OUT_PARAMS, SELECT_METHOD) and always follows the same communication protocol.

In case of derived classes, the state vector of an object is successively augmented by all attributes, which are newly defined in the derived class. Additionally, implementations for all methods, which are newly defined and refined have to be added, too.

When regarding polymorphic objects, the same method call can lead to the execution of different method implementations, dependent on the actual class membership of the object at invocation time. The class membership of a polymorphic object can be determined dynamically at system runtime, by means of a special tag, which is also part of each object's state vector. This tag holds the encoded information, to which class a certain object belongs to.

The following Figure 3 illustrates the structure being synthesized for a client/server relationship, if more than one client is concurrently accessing a server (a global object) as mentioned above. Concurrent accesses are scheduled by the arbiter belonging to the guarded object. The arbiter is connected to every client by an exclusive communication channel implementing an object's method interface.

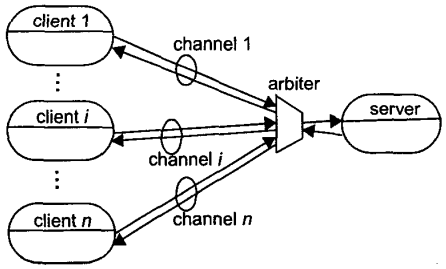

Fig.3. Client/server communication.

4.3 Optimisation

The synthesis of object-oriented hardware models without doing any optimisation tends to waste resources, especially in terms of circuit area and speed. For this reason optimisation of object-oriented structures is an essential task of the synthesis path developed in the ODETTE project.

The main approach for optimisation is to avoid the hardware implementation of redundant methods and attributes for each class instance. Due

to the fact, that classes are often being developed to be used in a wide range of different systems (reusability), a class may provide a broad set of different methods, from which only a small subset is actually used in a certain system or subsystem. Therefore the implementation of all the remainder methods would not make sense in that particular system. Especially polymorphic objects in connection with a distinct and deep inheritance hierarchy could cause an enormous resource overhead. The ODETTE synthesis tool which is now under development will perform various analysis to identify redundant methods and attributes and will then avoid their implementation in hardware.

5. CONCLUSION

In this paper we have presented an approach to introduce object-oriented modelling techniques into the development of hardware based on SystemC. The demonstrated synthesis techniques have already shown their practicability. The key aspects and problems of object-oriented hardware modelling and synthesis were presented and possible solutions were proposed. The development of a synthesis tool for object-oriented SystemC models has already started within the framework of the ODETTE project. Results of this project will allow to model hardware on a much higher level of abstraction without losing the benefit of automatic synthesis.

6. REFERENCES

[1] O. Levia, "Programming System Architectures with Java", IEEE Computer, August 1999.

[2] C. Passerone et al, "Modeling Reactive Systems in Java", ACM Transactions on Design Automation of Electronic Systems, vol. 3, no. 4, 1998, pp. 515-523.

[3] P. Schaumont, S. Vernalde, L. Rijnders, M. Engels, I. Bolsens, "A Programming Environment for the Design of Complex High-Speed ASICs", Proc. Design Automation Conference (DAC), 1998.

[4] K. Van Rompaey, D. Verkest, I. Bolsens, H. De Man, "Co-Ware - A design environment for heterogenous hardware/software systems", Proc. Euro-DAC, 1996.

[5] CynApps, Inc., "Cynlib, CynApps Class Library", information sheet, 1999.

[6] S. Maginot, W. Nebel, W. Putzke-Röming, M. Radetzki, "Final Objective VHDL language definition", REQUEST Deliverable 2.1.A (public), 1997; http://eis.informatik.uni-oldenburg.de/research/objective_vhdl.shtml

[7] P. J. Ashenden and P. A. Wilsey, "Principles for Language Extension to VHDL to Support High Level Modeling", Technical Report TR-03/97, Department of Computer Science, University of Adalaide, Australia; http://www.cs.adalaide.edu.au/users/petera/suave.html

[8] S.-T. Cheng, P. C. McGeer, M. Meyer, T. Truman, A. Sangiovianni-Vincentelli, P. Scaglia, "The V++ System Design Language", Proc. Design Automation and Test in Europe (DATE, Designer Track), 1998.

[9] M. Radetzki, "Synthesis of Digital Circuits from Object-Oriented Specifications", Dissertation at University of Oldenburg, 2000

[10] REQUEST; http://eis.informatik.uni-oldenburg.de/research/request.shtml

[11] Open SystemC Initiative; http://www.SystemC.org

[12] Synopsys, Inc., Frontier Design, Inc., CoWare, Inc., "SystemC Version 1.2Beta Users's Guide", 2001

[13] Synopsys, Inc., Frontier Design, Inc., CoWare, Inc., "Functional Specification for SystemC 2.0, Final, Version 2.0-M", 2001

[14] Synopsys, Inc.; http://www.synopsys.com

[15] C Level Design, Inc.; http://www.cleveldesign.com

[16] ODETTE; http://eis.informatik.uni-oldenburg.de/research/odette.shtml