Object-Oriented Modeling and Synthesis of SystemC Specifications

C. Schulz-Key, M. Winterholer, T. Schweizer, T. Kuhn, W. Rosenstiel
University of Tuebingen
Wilhelm-Schickard-Institute
Sand 14, 72076 Tuebingen, Germany
{oase,rosenstiel}@informatik.uni-tuebingen.de

Abstract— The constantly increasing complexity of today's systems demands specifications on highest levels of abstraction. In addition to a transition towards the system-level more elaborate techniques are necessary to close a growing productivity gap. Our solution to this problem is the application of the object-oriented programming paradigm together with the *de facto* industry standard *SystemC*. In this paper we show that this approach is feasible and present the integration of SystemC into a continuous object-oriented design flow. The design flow includes modeling with UML, hardware/software partitioning and synthesis of object-oriented specifications. We support our claim by results from a case study.

I. Introduction

As their complexity and size is constantly increasing, developing modern hardware/software systems is still a challenge. It is almost impossible to develop a system while adhering to the close time frame that is dictated by short product cycles.

This problem is usually addressed by making the specification on higher and higher levels of abstraction. State of the art in industry are behavioral descriptions on the register-transfer level that are well suited for monolithic designs.

The next higher levels of abstraction –the algorithmic-level and the system-level– are subject to ongoing research in the academic world. Especially the system-level is used for the specification of heterogeneous systems that consist of hardware and software partitions. Because of the high abstraction level the differences between the descriptions of the software and the hardware parts are getting smaller and smaller, and it is becomming feasible to describe the whole system within a single specification.

Using specification languages known from software development enables the designer to rapidly produce a *golden model* of the system that can be used for establishing its functional correctness. In an ideal case this model can then be used for automatically synthesizing the hardware parts of the system afterwards. To achieve this hardware capabilities are usually added to the software languages by extending the language or by providing special libraries.

Hardware/software co-design on the system-level is a very important step towards solving the productivity problem. Used alone, however, it is not sufficient: In order to master the main problem –the complexity of large systems– object-oriented paradigms additionally have to be applied which have shown their usefulness in the development of large software packages: By encapsulating elements with strong coherence and organizing data and behavior in one entity reuse is facilitated and the complexity generally is reduced.

For object-oriented specifications on the system-level with software languages two main variants have emerged: Java and C++. Java is very popular in academia [10, 22] as certain features, like the absence of pointers, make its analysis easier. C++ based approaches [9, 6] satisfy the mainstream, because of the widespread use of C++ for software development, and usually offer a high execution speed needed for fast simulation.

In the past years a number of C++ libraries for hardware specification have been developed. Notably the SystemC framework [17] has gained a huge popularity and its specification library seems to have become an industry standard.

Unfortunately, the object-oriented programming paradigm can not really be applied in conjunction with SystemC: Although the SystemC library itself is written in C++ using object-oriented techniques, the available synthesis tools do not allow object-oriented specifications. Current commercially available synthesis tools for SystemC can not handle those specifications and the designer has to manually integrate the source code of external objects and has to translate certain high-level constructs like threads and FIFOs. Virtanen et al. [20] list some design weaknesses in the SystemC library that restrict its use to procedural specifications. Another problem is the strong orientation towards hardware of SystemC which makes the specification of the software parts difficult.

In order to profit both from the benefits of SystemC and the benefits of object-oriented system-level specifications –without having to do the traditional manual translation– a more elaborate technique is necessary. In this article we therefore present a design flow that allows the synthesis of such specifications.

The remaining parts of the article are structured as follows: In section II an overview over existing approaches is given, followed by a description of some typical differences between traditional and object-oriented SystemC specifications in section III. After this, we present our design flow that allows modeling and automatic synthesis of object-oriented specifications. In sections V and VI we summarize the results of a case study. A conclusion is drawn in section VII.

II. Previous work

Using object-oriented languages to describe hardware is a wide spread solution for dealing with complex designs. Languages like SystemC and SystemVerilog v3.1 [1] are offering object-oriented constructs like classes and inheritance together with hardware specific features for algorithmic and RT level descriptions. C++ or Java based libraries like SystemC, PtolemyII [19] and Balboa [4] are available for modeling, using the object-oriented methodology to reduce the complexity of the designs.

Verification languages like Verisity's [21] *e* supported object-oriented specifications from the beginning but were primarily designed for facilitating verification. Capabilities for the synthesis of *e* were added later [11].

These approaches are showing that these new languages are well suited to describe hardware using object-oriented features. But the synthesis of these designs is still an area where a lot of work has to be done. Nowadays the refinement of a high level model into a synthesizable hardware description is a manual process. For a number of years Java is quite popular among researchers in the field of object-oriented

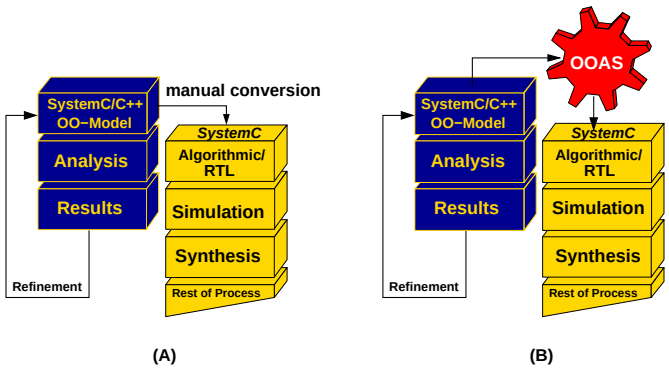


Fig. 1. Translations in the SystemC synthesis flow: (A) traditional, (B) automated using OOAS.

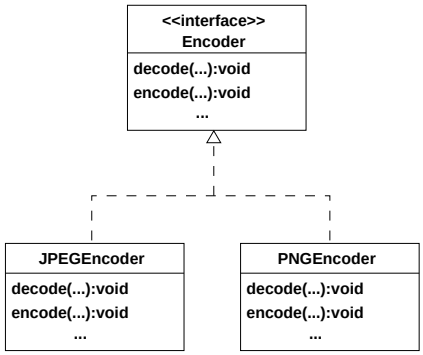


Fig. 2. As both classes implement the same interface they are mutually exchangeable.

synthesis [10, 22, 12]. In industry SystemC is more popular than Java, but is mainly used as a modeling language. Available synthesis tools like Cynthesizer [6] or CoCentric [16] do not allow the synthesis of object-oriented specifications [20] or limit its usage to a static subset. Grimpe et al. [8] propose an extension to SystemC (*SystemC-Plus*) to supplement the missing object-oriented capabilities, but there are still manual code changes required to use polymorphism or sharing objects between processes.

In the next sections we will describe the problems inherent in object-oriented SystemC synthesis and how they are solved by our approach.

III. Problems inherent in object-oriented SystemC synthesis

SystemC allows the description of the modules of a system on different levels of abstraction with the same language for the benefit of an iterative design flow. However, refining a module from a high-level functional specification down to an algorithmic- or RT-level model for synthesis (Fig. 1A) is usually a tedious manual and error-prone process. SystemC library classes, like `SC_FIFO` are not synthesizable — therefore they have to be translated manually into signals.

Often designs could be modeled more elegantly and re-use of already designed parts would be easier if user defined classes/objects could be inserted. Unfortunately user defined classes can not be synthesized and have to be translated manually. Due to the complexity of today's designs in most cases the manual translation is not feasible.

We identified the following main problems which have to be solved to synthesize an object-oriented hardware description: Object instantiation of classes, dynamic changes of referenced variables or pointers and polymorph method calls.

With our object-oriented analysis system (OOAS) we are addressing this problem (see section B.3). Using OOAS the automated synthesis of high-level SystemC models (Fig. 1B) becomes possible. In the next section we will describe our object-oriented design flow that uses a single specification for all parts of a system.

IV. Object-oriented design flow

In order to increase productivity we have to consider two basic rules for our design flow. The first rule implies that we apply established tools that have become a standard in industry whenever it is possible. The most important rule, however, is to hide the complexity of the system under development from the designers; this is achieved by following object-oriented design principles that have proven their usefulness during the last decade in the software development world.

Object-oriented synthesis allows system modeling on an abstraction level above the algorithmic level. Designers can express their ideas in a very natural way by thinking of classes rather than of data and procedures. Re-use is simplified by encapsulation. Together, this facilitates the development of complex systems. However, traditional hardware synthesis algorithms fail to handle object-oriented specifications for a number of reasons:

1. Object-oriented specifications usually show quite a dynamic behavior,
2. Concepts like object references, inheritance, or polymorphism can often only be resolved at run-time (concept of late binding).

In most cases it is therefore not straight-forward how to deduct the control data flow graph (CDFG) from the specification that is needed by synthesis tools.

Throughout the next paragraphs we will use a JPEG encoder to illustrate the steps of our design flow. The JPEG encoder is part of a digital camera. It is derived from an abstract class `Encoder`:

```
class JPEGEncoder :  Encoder { ...  }
```

The abstract class (or interface[1]) defines the exact signature for all classes that are derived from it (or that implement its interface). Therefore it is very easy to exchange the compression algorithm for the camera, as long as the class of the other algorithm implements the same interface (Fig. 2):

```
class PNGEncoder :  Encoder { ...  }
```

Now

```
Encoder *enc = new JPEGEncoder();
```

can be changed to

```
Encoder *enc = new PNGEncoder();
```

in a different version of the digital camera. This is the only change that is necessary in the source code of the camera specification!

Next we will show how the encoder is first modeled and refined, then analyzed and finally synthesized.

A. Modeling with UML

The first step in designing a system is creating an appropriate model. During this phase tasks are identified and responsibilities are mapped onto different classes. Several guidelines have been proposed how to capture the requirements and how to produce a specification from them. For more details we refer to the literature [3, 14]. Whenever possible previously developed classes are re-used and adapted to the new system using inheritance. The *Unified Modeling Language* (UML) [5] is used during this process to write the specification down

[1] An interface is a more general form of an abstract class. This notation is often used with Java.

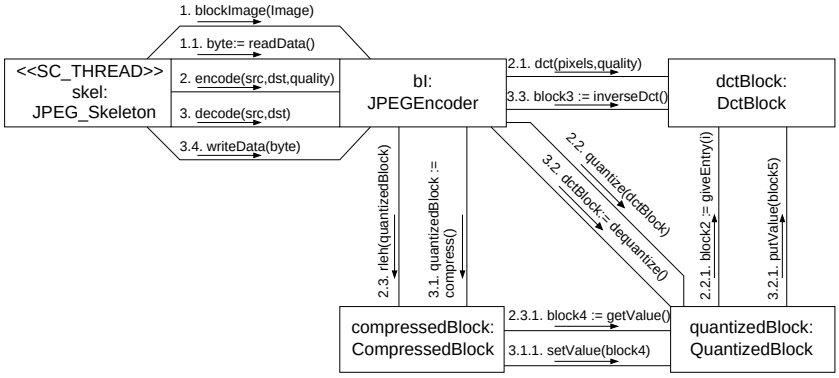


Fig. 3. Base for the case study: JPEG encoder collaboration diagram.

in a formal notation; tools like *Together* [18], *RationalRose* [13], or *StP* [2] facilitate this process of progressive refinement.

A.1 Refinement

If we graphically enter the class and collaboration diagrams of the JPEG encoder with one of those tools, the tool can generate the source code in the desired output language. For SystemC it was necessary to extend existing tools[2], as it is not well known in the software development world. We have written simple SystemC code generation modules for *Together* and *StP*. With them it is for example possible to automatically generate the following code for the constructor of the `SC_MODULE` that was marked with a `SC_THREAD` stereotype in the collaboration diagram of Fig. 3:

```
SC_CTOR(JPEG_Skeleton) {
   jpeg = new JPEGEncoder();
   SC_THREAD(do_run);
}
```

Besides such statements the generated code is a frame with the structure of the design and all its classes and methods. It guides the designer in refining the specification by implementing the algorithm in the methods.

Once the code for the algorithm is inserted by the designer, the code can be compiled. By executing the resulting binary and connecting it to external stimuli the functional correctness can be established. One of the advantages of using SystemC lies in its fast simulation that allows to speed up this important design stage. These steps have to be repeated until the desired results are obtained.

A.2 Partitioning

Because we are specifying a complete system, the specification has to be partitioned into hardware and software parts. In order to enable the designer to control the partitioning, it has to be expressed by means of the modeling language UML. For SystemC this means that all objects marked with a `SC_THREAD` or `SC_MODULE` stereotype and the objects aggregated to them are considered hardware. Before the specification is handed to OOAS for further analysis, the interfaces for the communication between the hardware and software partitions are generated by our tool *IFGen*. This step is necessary if method calls on components in different partitions should be allowed. In certain cases it is possible to use a special protocol for this kind of communication. The necessary ports and the code for the protocol can be generated automatically. For details we refer to [15].

[2]Most UML tools offer an interface that allows an extension.

B. Control Flow Analysis with OOAS

Traditionally hardware synthesis tools require a control data flow graph (CDFG) on which they can perform their transformations. Object-oriented specifications do not *a priori* have such a CDFG. Therefore one of the main tasks of our object-oriented analysis system (OOAS) is to deduct a CDFG from the information inherent in an object-oriented specification. To make this possible certain restrictions apply:

Due to the static nature of hardware it is not possible to dynamically allocate objects at *run-time*. Therefore OOAS rejects specifications that instantiate objects in loops with unknown number of iterations; the maximum number of instances of every object must be determinable at *compile-time*, when the specification is analyzed by OOAS. For the same reason the instantiation of objects is not allowed within cyclic or recursive method calls.

B.1 Input

The object-oriented input for the analysis can be written in a number of languages. OOAS can handle Java and *e*; support for C++ and SystemC was added only recently.

The first step of the analysis (Fig. 4) consists of the construction of an abstract syntax tree (AST) that is built by several parsers. After the AST has been built and the class hierarchy has been determined the previously marked hardware and software partitions (section A.2) are identified. For the software part no further analysis is required and it is written to a file in the C++ language. The hardware part is handed to the next step, the object control data flow analysis.

B.2 Resolution of object-oriented constructs

To capture all informations from an object-oriented specification in a single CDFG, constructs like classes, inheritance and polymorphism have to be resolved during a static object analysis. The main problem during this static analysis lies in the usage of references: Deciding which object is accessed when a method is called on a variable, is highly dynamic and can only be done at run-time. Objects may have several references, or aliases at the same time, therefore it is hard to tell which statements affect which object.

To illustrate this problem consider the class diagram in Fig. 2. The two encoding classes are implementing the same interface and as a result of that, the methods of those classes are polymorphic. A variable of type `Encoder` can reference an object of class `JPEGEncoder` or `PNGEncoder`. The target of this variable can change during runtime, referencing different objects of different classes. If the method `encode()` is called, it depends on the actualy referenced object which implementation of the method is chosen. In the following code

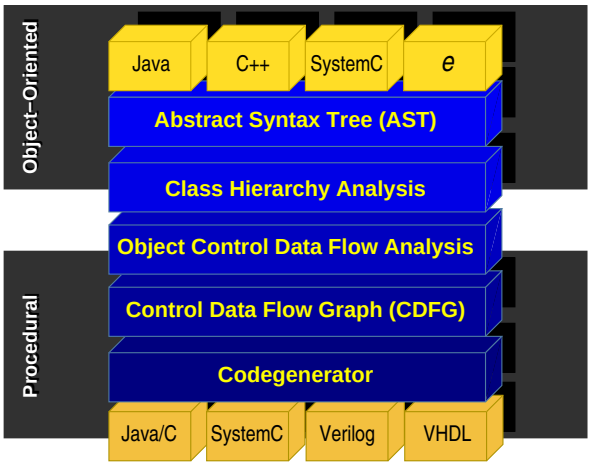


Fig. 4. Object-oriented Analysis System.

sequence, the variable `enc` references different objects depending on a condition which can't be resolved statically:

```
Encoder *enc;
if (cond == 0) {
  enc = new JPEGEncoder();
}
if (cond == 1) {
  enc = new PNGEncoder();
}
enc.encode();
```

To solve this problem, our analysis determines a set of possible objects for each variable within the scope of the currently analyzed statement. A scope table for each statement maps each variable of a scope to the objects that may be referenced by this variable. The scope tables are built during traversal of the control flow. At each branch of the flow, the set of references (*reference set*) is copied, and the sets are merged on merging points of the control flow.

A reference set $R_s(\texttt{enc})$ is the maximal set of all references on objects, which can be hidden by the alias `enc` after the execution of statement s under consideration of the type of the variable and the preceding control flow. All reference sets $R_s(\texttt{enc})$ of variables accessible in a CFG node n, build the scope table $S(n)$.

Modification of object references in the `if`-statement makes the call of the method on the variable `enc` ambiguous. Without considering the condition, the analysis merges the two possible targets for the variable after the `if`-statement.

The information from the reference set is used for the analysis of the method-call `enc.encode()`, where –depending on the conditional variable `cond`– the method `encode()` is called on an instance of class `JPEGEncoder` or `PNGEncoder`. For the analysis of the method body of `encode()`, all field accesses to the local object are assumed to change the data structure of the two possible objects. Alternatively the method can be in-lined for each object instance and the reference sets are reduced to two sets with one referenced object for each variable. In-lining of methods reduces the data dependencies but increases the code length. The resulting in-lined method-call can be seen in the following VHDL code excerpt:

```
if (cond  =  0) then
  enc := JPEGEncoder;
end if;
if (cond  =  1) then
  enc := PNGEncoder;
end if;
case enc is
  when PNGEncoder =>
    -- Inlining of PNGEncoder_1_encode_1
    ...
  when JPEGEncoder =>
    -- Inlining of JPEGEncoder_1_encode_1
    ...
  when others => null;
end case;
```

The analysis always terminates because of the constant number of objects in the whole system. The number of objects in a reference set can not exceed the maximum number of objects instantiated in the whole system. The constant number of objects is guaranteed since the instantiation of objects in loops is only allowed if the number of iterations can be determined at compile time. The instantiation of objects is not allowed within cyclic or recursive method calls.

The transformations shown so far apply to all object-oriented specification languages. Next we will describe a selection of SystemC specific constructs that need to be treated specially in order to make them synthesizable.

B.3 Transformations

The traditional SystemC design flow relies on the manual translation of certain high-level constructs. This adds to the complexity of the design that the developer sees, which we want to avoid; the manual translation is a source of errors, too. To improve this, OOAS can be used to automate the translation for certain constructs. As the results of the translation only show up in the generated output of OOAS –that is usually not necessary to be read by humans– the complexity of the design is not increased. OOAS can currently deal with the following constructs:

Modules: A `SC_MODULE` contains all methods which are started as processes in the constructor `SC_CTOR`. All threads started within the constructor are analyzed sequentially. A module can contain regular methods and ports, variables and references to user defined objects.

Threads: For each `SC_METHOD`, `SC_THREAD` and `SC_CTHREAD` a separate HDL process is generated. Reset behavior is supported through the watch list. An example of a module with one thread and the resulting generated Verilog code can be seen in Fig. 5

Standard data types: The standard data types of SystemC like `sc_logic`, `sc_lv`, `sc_bit`, `sc_bv`, `sc_int`, `sc_uint`, `sc_bigint`, and `sc_biguint` are mapped to the appropriate output types.

Ports: `sc_in`, `sc_out`, `sc_inout` are modeled as a HDL port or signal.

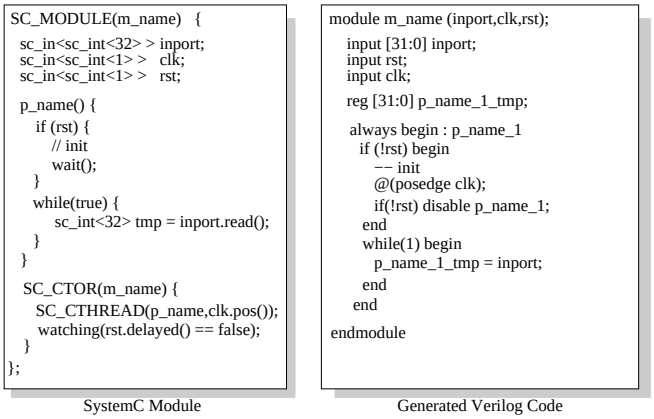
Events: `sc_events` are modeled as HDL signals.

User defined classes: User defined data-types and classes are automatically in-lined during the control flow analysis, as described above. Objects can be instantiated dynamically within the code, but not within infinitive loops or recursive method calls. If a class is instantiated multiple times, the field names of the object are extended with the class name and instance number as a prefix.

Channels: Currently we only support primitive channels, in the form of `SC_FIFO`. For the FIFO and its `read()`/`write()`-Methods code is generated that allows the communication via signals.

B.4 Output

Once the CDFG is established, it can be written out in one of several languages suitable for further processing. OOAS produces output in



```
SC_MODULE(m_name)   {
  sc_in<sc_int<32> > inport;
  sc_in<sc_int<1> >   clk;
  sc_in<sc_int<1> >   rst;

  p_name() {
    if (rst) {
      // init
      wait();
    }
    while(true) {
      sc_int<32> tmp = inport.read();
    }
  }

  SC_CTOR(m_name) {
    SC_CTHREAD(p_name,clk.pos());
    watching(rst.delayed() == false);
  }
};
```

SystemC Module

```
module m_name (inport,clk,rst);
   input [31:0] inport;
   input rst;
   input clk;
   reg [31:0] p_name_1_tmp;

   always begin : p_name_1
     if (!rst) begin
       -- init
       @(posedge clk);
       if(!rst) disable p_name_1;
     end
     while(1) begin
       p_name_1_tmp = inport;
     end
   end
endmodule
```

Generated Verilog Code

Fig. 5. A simple SystemC module and the automatically generated HDL code

procedural Java and C that can be used for analysis with other tools. For the synthesis of the specification, hardware description languages are used: The CDFG can be written out in procedural SystemC[3], VHDL, or Verilog on the algorithmic level, respectively. This step completes the transformations performed by OOAS.

C. Synthesis

The remaining parts of our object-oriented design flow can entirely be handled by existing tools. For the Verilog code and the VHDL code that is produced by OOAS the designer can choose from the synthesizable subset of several commercial high-level synthesis tools.

The extracted C++ code for the software partition of the system has to be compiled by a target compiler for the desired CPU core. Putting it all together we can now demonstrate the applicability of our object-oriented SystemC synthesis with a case study.

V. Case study

In order to measure the quality of the transformations performed by our tools we applied it to several benchmark tests. One of those tests is the JPEG encoder that already served as an example in the previous section. We implemented it in SystemC and compared the synthesis results with the results from a procedural VHDL implementation of the encoder.

The hardware part of the object-oriented implementation of our encoder consists of five classes whose collaboration diagram is shown in Fig. 3. The encoding algorithm performs many transformations involving large data structures containing the pixel data.

Because of the strong encapsulation of the objects it was very easy to re-use a previously developed class for the discrete cosine transformation (`DctBlock`) that we had in a non-SystemC specific form (i.e. as a user defined class).

The encoder was designed analog to the design flow described in section IV: First it was modeled together with the software part using the modeling tool *Together*[4]. The functional correctness of the implementation was established by compiling and executing this golden model.

In the next step this model was processed by our OOAS and synthesizable VHDL code was generated. The output was synthesized using a commercial high-level synthesis tool, aiming towards a clock period of 50 ns as it was done with the plain VHDL version. The results from the synthesis will be discussed in the next section.

[3]In the current version of OOAS only a very small subset of SystemC is generated as output language. Therefore we used VHDL as output language for the case study in section V.

[4]As current versions of *Together* can not cope with the C++ macros and templates that are heavily used with the SystemC library, we had to use a special mock-up version of the SystemC header-files that hides the macros during the modeling phase.

	VHDL		SystemC			
	area	delay	area		delay	
	[units]	[ns]	[units]	%	[ns]	%
JPEG	193 707	48.65	223 777	+15.5	48.80	+0.3
GCD	4 390	28.98	4 525	+3.1	27.63	-4.7
BINO	23 352	49.70	23 540	+0.8	49.69	$\pm$0.0
BSRT	1 837	14.20	2 404	+30.8	15.52	+9.3
FIBO	3 389	18.90	3 608	+6.5	19.14	+1.3

TABLE I
SYNTHESIS RESULTS OF OBJECT-ORIENTED SYSTEMC SPECIFICATIONS COMPARED TO THE RESULTS FROM EQUIVALENT PROCEDURAL VHDL VERSIONS

VI. Experimental results

The results from the synthesis of the object-oriented SystemC version of the JPEG encoder are within a quite close range to the results of the VHDL specification, as it can be seen in Table I. The tools were able to schedule the components within the given clock period of 50 ns. The area footprint of the object-oriented implementation is 15.5 % higher than the one of the hand-optimized VHDL implementation. It is composed of all the cell area and the net interconnect area. It is estimated by a commercial RT synthesis tool and given in units of the used *lca300k* library. A certain overhead in area consumption has to be expected for the code necessary for object handling and a more general port concept. It is relatively high for the JPEG example because of the very complex object structures, but usually varies from example to example:

The other benchmark tests –common denominator (GCD), binomial coefficient computation (BINO), the bubble-sort algorithm (BSRT) and the calculation of Fibonacci numbers (FIBO)– show interesting results: These are relatively small designs that do not benefit much from an object-oriented implementation; the part of the algorithm in the generated VHDL code looks very similar to the hand-coded VHDL version. Therefore the small overhead introduced by OOAS, that is mainly caused by the more general port concept, can be seen. The percentage of overhead of the bubble-sort implementation seems to be very high, because of the small total area of this benchmark; the absolute numbers do not look bad at all.

VII. Conclusions

In this paper we presented our approach for adding object-orientation to SystemC synthesis. The complete process starting from the object-oriented specification down to netlists is covered by our object-oriented design flow. We use established standard tools for this as much as possible by transforming the specification appropriately, no manual manipulations of the specification were necessary.

With our OOAS `SC_THREAD`, code with inheritance, polymorphism and user defined classes others than the classes defined in the SystemC library can be used for specification and are synthesizable. This way SystemC specifications can profit from the object-oriented programming paradigm: Re-use can be simplified and complexity can be reduced.

The standard SystemC design methodology relies on a manual transformation of system-level SystemC descriptions into synthesizable algorithmic- or RT-level models. This error-prone step can be automated with our approach. The integration of user defined classes

into the specification also allows the description of parts that are not necessarily as hardware specific, as they would be when they are encapsulated in the SystemC macros. Therefore the designer has more flexibility to decide whether a certain part should be instantiated in hardware or in software and the level of abstraction can be extended further to the system-level. To achieve this no extension of the SystemC language is needed, since everything necessary can be expressed with regular C++ statements.

An experimental validation with some benchmarks indicates the applicability of the OOAS flow for system synthesis. For most examples the results show some overhead caused by the automated translation. This price that has to be paid is quite small considering the benefits like better re-use and shorter design times that the application of object-oriented programming techniques has to offer. The integration of new optimizations to reduce the overhead for polymorhic method calls like the ones presented in the ODYSSEY project [7] will lead to even better results in near future.

ACKNOWLEDGMENTS

This work is funded by DFG project No. RO 1030/8-2 and Robert Bosch GmbH, Germany.

REFERENCES

[1] Accellera. SystemVerilog. http://www.systemverilog.org/.

[2] Aonix Inc. http://www.aonix.com/content/products/stp/uml.html.

[3] G. Booch. *Object-oriented Analysis and Design*. Addison-Wesley, 1994.

[4] F. Doucet, M. Otsuka, S. Shukla, and R. Gupta. An environment for dynamic component composition for efficient co-design,. In *Design Automation and Test in Europe (DATE'2002)*, 2002.

[5] B. P. Douglass. *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*. Addison-Wesley, Reading, MA, 1999.

[6] Forte Design Systems. Cynthesizer. http://www.forteds.com/.

[7] M. Goudarzi, S. Hessabi, and A. Mycroft. Object-oriented asip design and synthesis. In *Forum on Specification and Design Languages (FDL'03)*, Frankfurt, Germany, 2003.

[8] E. Grimpe and F. Oppenheimer. Object-oriented high level synthesis based on SystemC. In *ICECS 2001: The 8th IEEE International Conference on Electronics, Circuits and Systems*, volume 1, pages 529 – 534, St. Julian's, Malta, September 2001.

[9] T. Groetker, S. Liao, G. Martin, and S. Swan. *System Design With SystemC*. Kluwer Academic Publishers, May 2002.

[10] R. Helaihel and K. Olukotun. Java as a specification language for hardware-software systems. In *IEEE/ACM International Conference on Computer-Aided Design*, 1997.

[11] T. Kuhn, T. Oppold, M. Winterholer, W. Rosenstiel, M. Edwards, and Y. Kashai. A framework for object oriented hardware specification, verification, and synthesis. In *Proceedings of the Design Automation Conference (DAC'2001)*, 2001.

[12] T. Kuhn, C. Schulz-Key, and W. Rosenstiel. Object oriented hardware specification with Java. In *Proceedings of the ninth workshop on Synthesis and System Integration of Mixed Technologies (SASIMI'2000)*, Kyoto, Japan, 2000.

[13] Rational Software Corporation. http://www.rational.com/ products/rose/index.jsp.

[14] Rational Software Corporation. Rational Unified Process. http://www.rational.com/products/rup/index.jsp.

[15] C. Schulz-Key, T. Kuhn, and W. Rosenstiel. A framework for system-level partitioning of object-oriented specifications. In *Proceedings of the tenth workshop on Synthesis and System Integration of Mixed Technologies (SASIMI'2001)*, Nara, Japan, 2001.

[16] Synopsys Inc. CoCentric System Studio. http://www.synopsys.com/products/cocentric_studio/.

[17] SystemC. http://www.systemc.org/.

[18] TogetherSoft. http://www.togethersoft.com/us/products/.

[19] University of California, Berkeley. Ptolemy II. http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[20] S. Virtanen, D. Truscan, and L. Johan. SystemC based object oriented system design. In *Fourth International Forum on Design Languages (FDL'01)*, Lyon, France, September 2001.

[21] www.verisity.com.

[22] J. S. Young, J. MacDonald, M. Shilman, P. H. Tabbara, and A. R. Newton. Design and specification of embedded systems in Java using successive formal refinement. In *Proceedings of the Design Automation Conference (DAC'1998)*, 1998.