Java Based Object Oriented Hardware Specification and Synthesis*

Tommy Kuhn

Computer Engineering
University of Tübingen
72076 Tübingen, Germany
Tel: +49-7071-29-75940
Fax: +49-7071-29-5062
e-mail: kuhn@informatik.uni-tuebingen.de

Wolfgang Rosenstiel

Computer Engineering
University of Tübingen and FZI
72076 Tübingen, Germany
Tel: +49-7071-29-75482
Fax: +49-7071-29-5062
e-mail: rosen@informatik.uni-tuebingen.de

Abstract — In this contribution we show how the object oriented programming language Java can be used for the specification of synthesizable hardware. In this context the JavaBeans component model plays an important role. We show an integration of the JavaBeans model which allows the specification of hardware from different views and on different levels of abstraction. Furthermore we point out restrictions of the language necessary to perform high level synthesis.

I. Introduction

Recent years have brought a considerable increase in the complexity of integrated circuits which can only be handled by descriptions on higher and higher levels of abstraction. The complexity is transferred from modeling to synthesis and simulation. Thus, rapid simulation and, above all, reuse concepts become of decisive importance and should be supported by an appropriate description language. This language should be at least object oriented.

Quite many approaches are based on object oriented programming languages. One of the best known candidates is C++. It has its special advantages when it is used for simulation because of its high performance [3]. Also Java has recently gained more and more popularity in the field of hardware design [1, 2]. From the view of modeling and synthesis we also consider Java a suitable language.

Future systems will be characterized by embedded systems. These systems consist of a structural composition of hardware and software parts which are themselves specified on register transfer, algorithmic and system level. With a suitable language it must be possible to describe all parts of such systems. We therefore show how Java can be applied on different levels of abstraction and views. After that we present our design methodology. This methodology allows us to reduce simulation and synthesis effort significantly by supporting reuse techniques. Finally we will discuss the main synthesis problem.

*This work is partially supported by the DFG under RO1030/8.

II. Specification Process

A. Specification and Simulation Library

For the specification of hardware with object oriented programming languages class libraries play an important role. During specification it must be possible to express functionality on register transfer level as well as on algorithmic and system level. It must also be possible to use hardware specific data types and operations like bit vectors, ports or boolean operations, respectively. For this reason a class library has to provide all the necessary functionalities. Fig. 1 shows an example of such a class library. The main classes are RTHWO

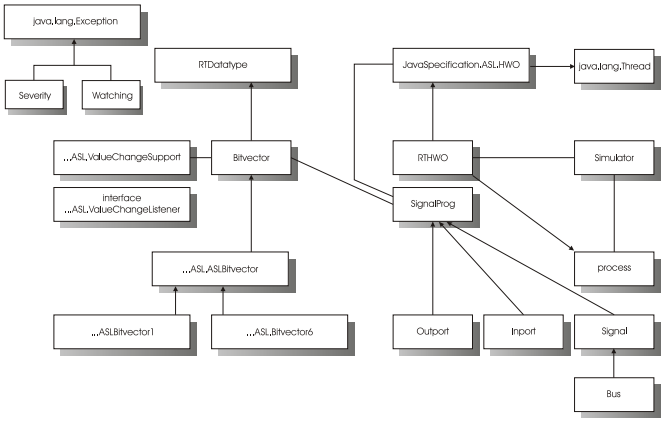

Fig. 1. Specification and simulation library for Java based hardware descriptions.

and HWO. By subclassing RTHWO a register transfer level component is specified. By subclassing HWO an algorithmic or system level component is specified. The ports of a component are specified in accordance to the JavaBeans component model with *set* and *get* methods. An input port is specified by a *set* method, and an output port by a *get* method. If both methods are given a bus is generated.
The width of operands is specified by using an appropriate bit vector from the given library or by extending the operands

identifier. The following code sequence shows how a circuit on algorithmic level is specified:

```
import JavaSpecification.ASL.*;
class DctBlock_Skeleton extends HWO {
   // Properties
   private final Bitvector2 function;
   private final Bitvector1 start;
   private final Bitvector1 ready;
   private final Bitvector1 valid;
   private final int op1 [];
   ...
   DctBlock_Skeleton() { // reset code
      function = new Bitvector2();
      valid    = new Bitvector1();
      ready    = new Bitvector1();
      ready.write(0); valid.write(0);
      ...
   }
   public void run() {
      // main method of thread
      ready.write(0);
      int func_2 = function.read();
      DctBlock obj1 = new DctBlock();
      switch (func_2) {
         case 1: obj1.dct(op1); break;
         ...
      }
   }
   public void setFunction
                     (Bitvector2 bv) {
      bv.cloneTo(function)
   }
   ...
```

Function is an input port. For simulation reasons the argument of a call to the corresponding *set* method copies the argument into an internal property. This technique simulates a call-by-value with the Java call-by-reference method calling mechanism in order to ensure a port like behavior of the interface. The local variable *func_2* demonstrates the possibility to declare variables of arbitrary bitwidth. The synthesis system explained in the next section recognizes that *func* is a variable of width 2. With the same technique a mapping of arrays into internal or external RAM components can be specified.
The distinction between classes inhereted from RTHWO or HWO allows to describe structure and behavior completely analogously. The classes can be treated identically and thus structure (RTHWO and HWO classes) and behavior (other classes) can be combined very easily. Furthermore, the event handling mechanism of JavaBeans allows to combine structural components by establishing connections graphically using one of the available visual builder tools.

B. Language Restrictions

No restrictions to the language are necessary in order to specify for simulation. However, for synthesis very few restrictions regarding the supported subset of the Java language have to be noted:

1. **Java Class Library.** Only a few selected classes of the Java Class Library are synthesizable. But for none of the classes it is reasonable to synthesize them because of their specified methods. For example, lets look at the `start()` method of a thread. It makes sense to use this method in a specification, but the contents of the implementation must not be used for synthesis. Therefore we introduce the concept of *semantic methods*. Our analysis and synthesis tool has a list of known *semantic methods*. If they are used in a specification they are treated like abstract methods, whose implementation will be substituted by the synthesis subsystem.

2. **Method calls.** Any method calls with any parameters are allowed. However the calls must not contain any cycles; particularly recursions are not possible.

3. **Instantiation.** Every instantiation of an object needs to be statically determinable. For this reason the generation of object instances within loops are forbidden. Dynamic data structures are also not possible.

4. **Correctness.** All specifications with Java have to meet the Java Standard and need to be compilable with an ordinary Java compiler.

III. SYNTHESIS

The main problem of synthesis from object oriented languages in general and from Java specifications in particular is handling of polymorphism. Polymorphism is one of the most important object oriented concepts. Due to polymorphism the control flow of an object oriented specification is not explicitly given but results from the values of variables while executing the program. Thus the main synthesis problem consists of the construction of a control flow graph.
In our approach we've developed a control data flow analysis technique to statically determine all possible references variables can have during runtime. This technique was extended by an iterative approach to also analyze data and control dependencies between concurrent threads in system level specifications. In a first phase all threads of a given specification are analyzed iteratively. With the results of this analysis phase each thread can be rewritten as a control flow graph.
Now its possible to print out the control flow graphs in different languages. All objects are represented by symbolic values. Polymorphic method calls are resolved using switch/case-statements. Fig. 2 gives a brief overview of the system. The control flow graphs are rewritten in a procedural kind of Java (Java without any objects) and behavioral VHDL. This allows to verify the correctness of the analysis and transformation step and to use commercially available high level synthesis systems to produce the desired circuit. Each control flow graph

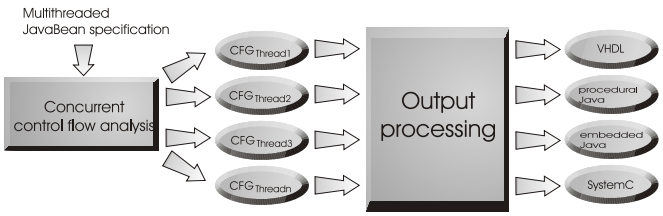

Fig. 2. Control flow generation and output processing for multithreaded JavaBeans.

is transformed into its own VHDL process. All the synchronization mechanisms are mapped onto sequences of semantically equivalent VHDL language constructs. Variables written by different threads are mapped onto VHDL signals. Write access to these signals is given by a bus arbiter component [4] which is inserted into the resulting hardware. The same component (with additional logic, of course) is used to realize the semantic of *synchronized* methods in conjunction with *wait/notify* method calls.
Beyond it, the control flow graphs can also be rewritten for example in C++ (e.g. using the SystemC library [3]). This allows fast simulation by execution while preserving Java as a specification language.

IV. Experimental Results

We've specified several circuits in Java (e.g. JPEG codec, fuzzy controler, GSM speech transcodec, etc.). They were processed and transformed to VHDL with the system mentioned above. The results of the high level synthesis showed that the area of the circuits increases by approximately 5% compared to hand written VHDL code. The delay estimated by a commercial register transfer and logic level system increases by approximately 15%.

V. Summary and Conclusions

In this contribution we presented a class library in order to use Java for the specification of hardware on different levels of abstraction and from different views (e.g. structural and behavioral). Along with the integration of the JavaBeans component model it is possible to specify structure as well as behavior as well as a mixture of structure and behavior with commercially available visual builder tools.
We also gave a brief overview over our synthesis approach. A static data flow analysis technique allows to generate control flow graphs from multithreaded JavaBean specifications. They can be rewritten in different languages (e.g. VHDL, C++ etc). Thus a very powerful system is available wich allows to use Java as a specification language without having to abandon the benefits of other languages (e.g. fast simulation with C++).

References

[1] Helaihel, R. and Olukotun, K., "Java as a Specification Language for Hardware-Software Systems", *Proceedings of ICCAD*, 1997.

[2] Young, J. S., MacDonald, J., Shilman, M., Tabbara, P. H. and Newton, A. R., "Design and Specification of Embedded Systems in Java Using Successive, Formal Refinement", *Proceedings of DAC*, 1998.

[3] Synopsys, Inc., "SystemC", Reference Manual Release 0.9,1999.

[4] Madsen, J. and Brage, J. P., "Modeling Shared Variables in VHDL", *Transactions on the ACM*, 1994.