Specification and validation of distributed IP-based designs with JavaCAD

Marcello Dalpasso
DEI – Università di Padova
Via Gradenigo, 6/A – 35131 Padova, Italy
dalpasso@dei.unipd.it

Alessandro Bogliolo, Luca Benini
DEIS – Università di Bologna
Viale Risorgimento, 2 – 40136 Bologna, Italy
{abogliolo,lbenini}@deis.unibo.it

Abstract

This paper presents JavaCAD, a new Java-based CAD framework for the design, validation and simulation of systems using third-party components with reciprocal intellectual property (IP) protection. The designer can use remote components with a dedicated and secure Internet protocol, that guarantees IP protection and supports a smooth transition between component evaluation and purchase.

1 Introduction

The increasing complexity of VLSI designs can be tackled only relying on design methodologies that emphasize the re-use of previously developed hardware components. In this line of evolution, the design of a large digital system will be centered on the selection, evaluation and acquisition of *intellectual property* (IP) components designed by others, and the construction of dedicated interfaces for their communication. A new generation of CAD tools is required to address the issues raised by the widespread diffusion of IP-based designs. In this paper we propose a new computer-aided design environment that has been conceived with the specific purpose of enabling and fostering IP-based design.

One of the main implications of IP-based design is the greater need for communication of valuable information between IP providers and users. The usefulness of design environments that facilitate communication and data exchange has been recognized in the past [16]. More recently, the explosive growth of computer networking in general, and the World-Wide Web in particular, has spurred the development of networked (or web-based) EDA tools and tool frameworks [8, 1, 10, 13, 7, 2, 4, 3].

In this work, we focus on the problems raised by the use of IP components provided by third-party vendors over the Internet. We envision the following scenario. A designer (IP-user) is potentially interested in employing an IP component developed by a third party (IP-provider). Before purchasing the IP component, she/he wants to verify that the component complies with the specification of the design under development. Such specifications may be functional or may constrain design cost metrics such as speed, area, power or testability. Ideally, the IP-user would like to be able to perform accurate simulation/evaluation of the IP component within her/his design, estimate its cost and finally decide on the purchase. On the other hand, the IP-provider cannot promptly disclose the complete information to the IP-users for several reasons. First, his intellectual property would be made available to the user at no charge. Second, for the purpose of evaluation, the provider may want to make sure that the user is deploying the component in a correct fashion, so that comparison with other competing products would be fair.

Our Java-based simulation environment, called *JavaCAD*, provides a solution to safe and effective IP evaluation during design specification/exploration. In the JavaCAD framework a designer can instantiate third-party IP components and simulate them together with proprietary blocks in a seamless fashion. Multiple IP components from multiple IP providers can be simulated and the protection of intellectual property is guaranteed for all providers involved; the intellectual property of the user with regards to the providers is protected as well.

We provide methods for enabling the negotiation between IP user and provider about the amount of information that can be made available during simulation. Thus, JavaCAD supports a smooth transition between evaluation and purchase.

JavaCAD is built around a flexible event-driven simulation engine, that supports hierarchy, multiple levels of abstraction, distributed and parallel simulation. Designs are specified directly in Java using a mixed structural-behavioral style that is widely supported in common hardware description languages such as Verilog and VHDL.

The paper is outlined as follows: the next Section ac-

counts for the most significant previous efforts in this field; Section 3 introduces the main features of the JavaCAD design framework and gives some details about its architecture; Section 4 outlines a scenario for using JavaCAD in a distributed, Web-based design environment that involves IP-users and third-party IP-providers; Section 5 concludes the work.

2 Previous work

Improving data communication among design tools and providing a uniform user interface were the main goals of several *framework* initiatives [16]. The definition of standard formats for design descriptions such as VHDL and EDIF has been another important milestone in this direction. With the explosive growth of networked computing new paradigms have emerged for organizing the access to design-specific information [10, 13, 11]: geographically distributed information can be automatically collected and visualized by exploiting the World-Wide Web. Web servers for management and diffusion of design-critical information have been developed by all major companies in the semiconductor business, as well as by EDA companies.

A more radical paradigm shift has been proposed [7, 2, 4, 3]: the World-Wide Web can be exploited not only to share information, but also to actually perform design-critical tasks, such as validation and optimization. In this more aggressive view, a designer is a *client* in a client-server architecture and he/she can request services from a variety of *servers*. CAD tool usage is a type of service, as well as intellectual property delivery. More recent research initiatives [4, 3] try to push the concept of client-server interaction and collaborative, distributed design even further.

Similarly to previous work, we assume a client-server architecture where clients are IP users and servers are IP providers. On the other hand, we focus on IP-related issues, more than trying to provide a common framework for general EDA-related interactions. We assume that hardware designs are specified in Java. Java-based system specification styles have been studied in [14, 6]. These works have assessed the suitability of Java as an abstract hardware-software specification language.

3 JavaCAD: Functionality and Architecture

The JavaCAD framework for system design, validation and simulation, has been developed entirely in Java [5]. We exploited the object-oriented features of Java to effectively express the component-based design semantics of modern digital systems. The key feature of JavaCAD is the capability of handling designs containing "local" (owned by the designer) and "remote" (provided by a third-party developer) components. Any design and simulation tool can exploit JavaCAD protocols (based upon JavaSoft Remote Method Invocation – RMI [12]) to seamlessly deal with remote components as if they were local. The hardware description language (HDL) for system designs used in JavaCAD is the Java language itself. The framework includes full support for hierarchical design at multiple levels of abstraction.

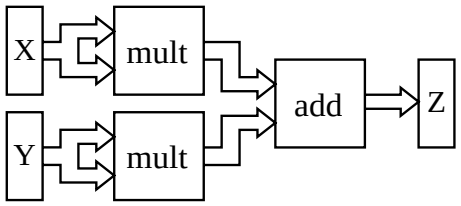


Figure 1. A simple RTL circuit that computes the sum of squares of two input words, X and Y, and stores the result in register Z.

The JavaCAD framework has been designed as a set of Java packages that must be used both by the IP provider and by the IP user: such packages contains the super-classes of any actual component developed by the user as well as by the IP provider, and all the utility classes that are required for the JavaCAD framework to work.

JavaCAD contains a multi-level event-driven simulator. The choice of the simulation engine was motivated by the generality and flexibility of event-driven simulation, as opposed to more specialized simulation techniques that target high efficiency at the expense of generality. Furthermore, JavaCAD has direct and full support (based on Java threads) for concurrent simulations of different parts of a design or of different setups for the same design.

Design specification requires the instantiation of JavaCAD classes that describe design components taken either from the standard JavaCAD packages, or from a user-designed library, or from third-party IP libraries. Design validation and simulation is performed by running the main JavaCAD class that implements the whole design under development.

Example 1 *Consider the simple RTL circuit shown in Figure 1. Assume that the designer (IP-user) wants to use a high-performance proprietary multiplier developed by an IP-provider. Before purchasing the multiplier, the IP-user wants to instantiate the IP multiplier in the design to verify its functional correctness. Furthermore, the IP-user wants to accurately estimate the register-to-register propagation delay to set the clock period.*

The IP-provider releases (through a JavaCAD server) the public interface of the multiplier, containing an abstract functional model (i.e., $out = in_1 * in_2$). The public interface allows the designer to perform functional simulation.

However, timing analysis requires the knowledge of the detailed gate level structure of the multiplier (which belongs to the not-yet-disclosed IP). The designer can perform timing analysis using the RMI features of JavaCAD, by calling a method that traverses the netlist of the multiplier. The method is run on the IP-provider server and only the final result is communicated to the IP-user. Notice that the adder and registers are local components. Their methods for timing analysis are invoked in the same way, but they are run locally on the IP-user machine.

3.1 Basic data structures

Any design component in JavaCAD is a subclass of the `Module` class. The `Module` object is specialized by: *i)* a set of methods that are executed when events reach the component (for instance, methods specifying functionality and cost metrics), *ii)* a set of connections, each one tied to a different `Connector`.

The *behavior* of the component can be specified at different levels of abstraction: gate and register-transfer level are actually supported in JavaCAD, but higher levels are planned and under development. As detailed in Section 3.3, a `Module` instance can be either local or remotely accessed through the Internet on the IP-provider server.

A `Connector` ties two `Module`s together, and it performs no other function but `Token`-passing between `Module`s (see Section 3.2); the connection of `Module`s can exist only through a `Connector`, thus it is easy to enforce any communication semantics between `Module`s by means of a custom `Connector`. For instance, bit-level and word-level `Connector`s are provided by JavaCAD to handle gate-level and word-level variables respectively (please note that *word-level* variables can have any word length).

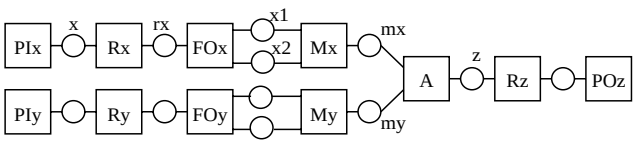
Since `Connector`s represent point to point connections, special `Module`s are needed to handle multiple fan-out nets and represent net delays. This gives the designer a high degree of flexibility: for instance, custom `FanOut` modules can provide different delays for the propagation towards different target `Connector`s.

In a similar way, custom `Module`s can be designed to handle an interface between a part of the design that is specified at the gate level, and another part that is described at the register-transfer level.

As an example of using Java as an HDL in JavaCAD, Figure 2 shows a class that describes the circuit given in Figure 1.

3.2 Simulation & Event Handling

The event-driven simulation engine of JavaCAD takes care of everything is needed to perform the actual validation and simulation of a distributed IP-based design, as well as to




```
public class Example extends Design {
   public void design() throws Exception {
      int width = 16;
      Connector x = new WordConnector(width);
      Connector rx = new WordConnector(width);
      ...

      Module PIx = new PrimaryInput(x);
      Module Rx = new Register(x, rx);
      Module FOx = new Fanout(rx, x1, x2);
      Module Mx = new RemoteMult(x1, x2, mx);
      Module A = new Adder(mx, my, z);
      ...

      Circuit C = new Circuit(PIx, Rx, FOx, Mx,
                              A, ....);
      Simulator s = new Simulator(c);
      ...
      s.start();
   }
}
```

Figure 2. The JavaCAD specification of the simple design of Figure 1. Squares and circles in the block diagram denotes `Module`s and `Connector`s respectively. The Java code is partially reported for the sake of conciseness.

perform the negotiation of parameters and the configuration of parameter estimators in the simulation set-up. The superclass for any *event* is the `Token`, and `Token`s are handled (i.e., scheduled and delivered) by a `Scheduler`.

It is possible to instantiate multiple `Scheduler`s and to run them in concurrent threads, thereby providing full support for concurrent simulations running over the same design. During simulation, internal state information for each component is stored in lookup tables addressed by unique identifiers associated with the `Scheduler`s.

Notice that `Token`s do not represent only functional events (i.e., changes of signal values), but they are used as a general communication method to traverse the design, collect information from modules, set up run-time parameters etc. In other words, `Token`s implement a general message-passing engine for design manipulation.

3.3 Remote Method Invocation (RMI)

Java Remote Method Invocation (RMI, [12]) is a JavaSoft implementation of a CORBA-like distributed object model

for the Java language that retains the semantics of the Java object model, making distributed objects easy to implement and use. The key features of RMI exploited by JavaCAD are:

- creation of local instances of remote classes (`Module`s) without having their byte-code available (such classes are IP components);
- invocation of methods of remote classes, with a proper handling of parameters and return value;
- handling of a secure Internet communication channel between the client (the IP-user) and the server (the IP-provider).

3.4 Security

Security is a key issue in JavaCAD, and guaranteeing security poses additional problems as compared to traditional security issues in distributed Internet-based client-server computing [15]. In fact, not only the communication between the client and the server must be secured from third-party intrusions, but even the client and the server cannot completely trust each other (for IP-protection goals).

Both these problems have been solved in JavaCAD by exploiting RMI features, and through an innovative use of the enhanced security model that comes in JDK 1.2 [9]. Traditional communication security over the Internet is completely handled by the RMI protocol, that uses sockets for communication; such sockets are customizable and can be as secure as the client and the server want and agree upon (e.g., SSL, DES, RSA).

3.4.1 Security as protection of the IP provider

Security for the IP provider has been implemented in JavaCAD by splitting the actual component specification in two main parts:

- the IP-protected part of the component specification is located on the IP-provider server as a private class, whose byte-code is not sent to the client even during the actual simulation;
- the public part of the component specification is given to the IP-user by the IP-provider and is used to instantiate the remote component within the actual design. Such a public part (an RMI *stub* [12]) handles the invocation of remote methods to be run on the IP-provider server.

3.4.2 Security as protection of the IP user

Security as protection of the IP of the designer that uses remote components has been implemented in JavaCAD by a careful usage of parameter marshalling in RMI (*Object Serialization*, [12]).

Bounding each `Module` with `Connector`s allows the JavaCAD framework to completely inhibit the transmission of sensitive information (the other actual `Module`s used in the design as well as their properties and mutual relationships). Since the remote IP-component (a `Module` itself) needs only information that are available at its own `Connector`s to perform any possible estimation and simulation, only such information is transmitted over the RMI channel.

In addition, the downloaded public and stub classes that make up the package supplied by the IP provider has no permission when executing in the designer environment, since they are marked as "non trusted" and consequently handled by the Java security manager.

4 Using JavaCAD

The JavaCAD framework consists of a set of Java packages (distributed as a JAR file) that must be used both by the IP-user and by the IP-provider.

4.1 The IP-provider perspective

An IP-provider wishing to use JavaCAD to promote the usage of its custom components without complex non-disclosure agreements must use Java as an HDL to write its `CustomModule` class, by subclassing the `Module` root class outlined in Section 3.1. Such a `CustomModule` implements the full functionality and support for simulation and validation that is available for the component, with no problem for the IP protection, since its IP-sensitive methods reside on the provider server only.

To make the component available to remote designers for evaluation, the IP provider should set-up a public Web server with pages that describe the key features of its component, and a link to JAR files for the download of the public and the stub parts of its component (*i.e.*, the necessary classes for an IP-user to instance the remote component in her/his design).

In addition, the IP-provider must set-up an RMI registry server and an RMI daemon (see [12]) where the IP-protected part of its component is active and ready to reply to incoming requests for estimation, simulation and validation, by means of the JavaCAD protocol, inclusive of negotiation capabilities as well as full logging of client connections and activities.

It is important to mention that RMI does raise performance issues. If remote methods are called very often, simulation performance will be bound by the limited (and unpredictable) bandwidth of Internet connection. For this reason, remote methods should be used sparingly, either to

process rare events or to perform static once-for-all analyses, and they should be concerned with the real protection of the IP only. It is responsibility of the IP-provider to design remote methods that limit performance losses.

Referring to the example of Figure 1, it would not be a good choice to implement the procedure that computes the results of the multiplication as a remote method. Such method is called during simulation every time an event arrives at the inputs of the multiplier, and it must complete execution before simulation can proceed. In addition, such a procedure does not carry any intellectual property, since the multiplier functionality can be considered public.

On the other hand, the timing analysis method is generally called a limited number of times, and its performance impact is much smaller. Such a method should be really remote, since its evaluation needs the gate-level description of the multiplier, that is the actual IP that must be protected.

4.2 The IP-user perspective

The designer using JavaCAD develops his/her circuit specification as an interconnection of `Module`s, freely mixing local components and remote IP-protected components using Java as an HDL. During validation and simulation of the design, the access to remote IP-protected components is transparently handled by the JavaCAD framework, possibly interacting with the user when money is required by the IP-provider for returning information.

Typically, the user should have access to the Web servers of different IP-providers, select the actual components to use in her/his design, download their public and stub part as JAR files, and instance such remote `Module`s. When the simulation set-up is started by the designer, the JavaCAD protocol establishes the Internet communication with the IP-provider, while ensuring the IP protection for both the designer and the provider.

After successful validation and simulation of the design, the IP-user's company can proceed to standard commercial relationship to use the IP-protected component, but no previous agreement was necessary.

5 Conclusions and future work

We have presented a new framework for system design, validation and simulation targeting IP-based design flows. Java Remote Method Invocation is used to guarantee IP protection while allowing the designer to perform accurate simulation of her/his design before actually purchasing (some of) the involved IP components. JavaCAD supports hierarchical design, multiple levels of abstraction, distributed (on the Internet) and parallel simulation. The Java language is directly used as a hardware description language.

Future and undergoing developments will address higher levels of abstraction (currently, gate and register-transfer level are supported) and flexible simulation setup with interactive client-server negotiation of simulation parameters.

References

[1] A. Bedenfeld and R. Camposano. Tool integration and construction using generated graph-based design representation. *Proc. of the Design Automation Conference*, pages 94–99, 1995.

[2] D. Lidsky and J. Rabaey. Early power exploration - a World Wide Web application. *Proc. of the Design Automation Conference*, pages 27–32, 1996.

[3] F. Chan, M. Spiller and R. Newton. WELD - An environment for Web-based electronic design. *Proc. of the Design Automation Conference*, pages 146–151, 1998.

[4] H. Lavana, A. Khetawat, F. Brglez and K. Kozminski. Executable workflows: a paradigm for collaborative design on the Internet. *Proc. of the Design Automation Conference*, pages 553–558, 1997.

[5] J. Gosling, B. Joy and G. Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[6] J. Young et al. Design and specification of embedded systems in Java using successive, formal refinement. *Proc. of the Design Automation Conference*, pages 70–75, 1998.

[7] L. Benini, A. Bogliolo and G. De Micheli. Distributed EDA tool integration: the PPP paradigm. *Proc. of the International Conference on Computer Design*, pages 448–453, 1996.

[8] L. Geppert. IC Design on the World Wide Web. *IEEE Spectrum*, June 1998.

[9] L. Gong. *The Java Security Model and Architecture*. Addison-Wesley, *announced*.

[10] M. J. Silva and R. H. Katz. The case for design using the World Wide Web. *Proc. of the Design Automation Conference*, pages 579–585, 1995.

[11] M. Spiller and R. Newton. EDA and the Network. *Proc. of the International Conference on Computer-Aided Design*, pages 470–475, 1997.

[12] P. Chan. *The Java Developers Almanac*. Addison-Wesley, 1998.

[13] P. G. Ploger et al. WWW Based structuring of codesigns. *Proc. of the International Symposium on System Synthesis*, pages 138–143, 1995.

[14] R. Helaihel and K. Olukotun. Java as a specification language for hardware-software systems. *Proc. of the International Conference on Computer-Aided Design*, pages 690–697, 1997.

[15] S. Hauck and S. Knoll. Data security for Web-based CAD. *Proc. of the Design Automation Conference*, pages 788–793, 1998.

[16] T. J. Barnes et al. *Electronic CAD frameworks*. Kluwer Academic Publishers, 1992.