IP Delivery for FPGAs Using Applets and JHDL

Michael J. Wirthlin and Brian McMurtrey
Brigham Young University, 459 CB
Provo, UT 84664
wirthlin@ee.byu.edu,mcmcurtre@ee.byu.edu

ABSTRACT

This paper introduces an FPGA IP evaluation and delivery system that operates within Java applets. The use of such applets allows designers to create, evaluate, test, and obtain FPGA circuits directly within a web browser. Based on the JHDL design tool, these applets allow structural viewing, circuit simulation, and netlist generation of application-specific circuits. Applets can be customized to provide varying levels of IP visibility and functionality as needed by both customer and vendor.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*Simulation, Hardware description languages*

General Terms

Design

Keywords

Intellectual Property, JHDL, Applet, FPGA

1. INTRODUCTION

The rapid growth in the density of Field Programmable Gate Arrays (FPGAs) has prompted the use of third-party intellectual property (IP) in large FPGA designs. Many vendors now offer a variety of design cores targeted specifically for one of many FPGA technologies. The advantages of IP-based design are well known – using third-party IP can reduce the design time, improve the design quality, and facilitate the design of systems too large to design without pre-designed cores.

While the use of third-party cores may improve design productivity, the process of delivering the IP from a vendor to a customer is not straightforward. The IP delivery process must balance the conflicting demands of IP visibility for the customer and IP protection for the vendor. For the customer, visibility is necessary to properly evaluate and validate the IP. Such visibility may require simulation of the IP, structural understanding of the IP, or some form of detailed IP characterization. The more visibility available to the customer, the more confidence he or she has that the IP operates as specified. Further, greater visibility eases the ability to integrate the IP within the customer's design. For the vendor, such customer needs conflict with the need to protect the IP from unauthorized use or duplication. If too much of the design is exposed to the customer, unauthorized use or reproduction may occur and significantly reduce the value of the IP. Successful IP delivery systems must balance the conflicting needs of user visibility and vendor IP protection.

The approach for IP evaluation and delivery described in this paper involves the use of Java applets executing within a user's web browser. A potential user may evaluate a given FPGA circuit by accessing a web page and interacting with the applet that executes within their local browser. IP evaluation applets can be written to provide various levels of both IP visibility and protection.

1.1 IP Evaluation and Delivery Using Applets

Java applets are executable binaries that operate *within* a web browser when accessed by the user. When the user accesses a web page containing an applet, both the `html` code used to render the web page and the Java binaries are downloaded by the browser. The browser then executes the Java binaries within a local Java Virtual Machine and displays the applet within the browser window. Java applets have been written for a wide variety of applications and web interfaces.

Custom applets can be used as a convenient and powerful way to evaluate and deliver electronic intellectual property on-line. Based on the user's license, a custom applet is presented that offers the appropriate IP evaluation and delivery functionality. Such applets may provide IP characterization, simulation, viewing, and other features as necessary.

Providing IP delivery executables in the form of applets has several advantages over the delivery of user-installed software. First, users do not have to install the software used for IP delivery or evaluation. Providing the executables online simplifies the process for the user and allows the potential customer to evaluate the IP with less effort. Second, providing the executable on the web server insures that the customer accesses the most recent code every time the associated web page is accessed. The online executables can be updated to reflect improvements and bug fixes by the vendor. Rather than redistributing the IP executables every

time problems are resolved, vendors need only update executables found on their online server. Third, it is relatively easy to control the executables accessed by the user. Based on user profiles, the web server can provide an executable applet customized to the needs or license of the user. Such user-specific executables will vary the level of observability and control provided to the module generator applet.

The applet delivery mechanism described by this paper uses the JHDL design environment – a design tool written in Java for creating FPGA circuits[4]. This tool allows designers to create, simulate, and netlist FPGA circuits by writing Java programs that instance appropriate FPGA library cells and wires. While JHDL has been used primarily as a stand-alone design tool, it offers a number of tools and aids for describing, evaluating, and delivering FPGA IP over the internet.

1.2   Related Work

A variety of techniques for web-based IP evaluation and delivery have been proposed and demonstrated. Similar approaches to the JHDL applet approach described by this paper will be introduced and contrasted below.

1.2.1   Web-CAD

The Web-CAD methodology provides the infrastructure for simulating IP over the internet using a client-server architecture[2]. Using the internet to communicate simulation information, this methodology allows users to evaluate IP *without* actually acquiring simulation models. In this system, protected IP is simulated at a remote server where the details of the simulation model are hidden from the user. Users can simulate protected IP within their simulation environment by transferring simulation information with the remote server using a publicly available socket library.

Unlike the Web-CAD methodology, the JHDL applet approach does not transfer simulation events across the network. Instead, a protected IP executable is sent over the internet in the form of a Java applet. Simulating the IP directly on the user's machine will result in increased simulation speed by avoiding the relatively long latency associated with a network.

1.2.2   JavaCAD

Like the Web-CAD methodology, the JavaCAD project provides an infrastructure for simulating and evaluating circuits over the internet using remote method invocation (RMI) [1]. This system also provides the capability of delivering detailed area, power, and delay estimates of the IP in question. Like the JavaCAD project, the IP presented in JHDL Applets is described in Java and simulated using a custom simulator. Unlike JavaCAD, the JHDL approach uses the network to deliver IP applets rather than transfer simulation data over the internet.

1.2.3   JBits

The JBits tool developed at Xilinx is another Java-based tool used for delivering FPGA IP cores [5]. This tool delivers pre-placed IP cores by *modifying* the configuration bitstream of the user. Because the IP is delivered in the form of changes to a proprietary configuration bitstream, the structure of the IP is hidden from the user. While this tool currently does not use the internet to deliver IP cores, JBits could easily be incorporated into online Java applets.

1.3   Overview

This paper will begin by providing an overview of the JHDL design environment and discussing its ability to describe FPGA intellectual property. Next, the notion of IP "executables" will be introduced along with an example of such an executable using JHDL. The use of JHDL within Java applets will be discussed and a detailed example of a JHDL applet for IP evaluation and delivery will be provided. Next, various scenarios for IP delivery applets will be suggested and the paper will conclude with directions for future work.

2.   JHDL

JHDL is a open-source design environment developed at BYU used for creating high-performance FPGA designs[3, 4]. Based on the Java programming language, users create FPGA designs by writing Java programs that instance FPGA components and wires found within the supported JHDL libraries. These Java files are compiled using conventional Java compilation tools and simulated within the Java Virtual Machine (JVM). The circuit is described using conventional Java programming constructs and class objects are "constructed" by instancing the object within a JHDL hardware system object.

The Java code shown below demonstrates the construction of a simple full-adder within JHDL using the and2, or3, and xor3 library elements:

```
public FullAdder(Node parent, Wire a, Wire b,
                 Wire ci, Wire s, Wire co) {
   ...
   Wire t1 = new Xwire(this,1);
   Wire t2 = new Xwire(this,1);
   Wire t3 = new Xwire(this,1);
   new and2(this,a,b,t1);
   new and2(this,a,ci,t2);
   new and2(this,b,ci,t3);
   new or3(this,t1,t2,t3,co);/* co is carry out */
   new xor3(this,a,b,ci,s);  /* s is output */
}
```

This example begins by constructing three intermediate one-bit wires: t1, t2, and t3. The carry out logic is created using three and2 primitives and the or3 primitive (i.e. co=a&b | a&ci | b&ci). Finally, the sum logic is created using a three-input exclusive or gate.

A variety of design aids are available for circuits constructed within JHDL. These tools can be used for manipulating, viewing, and simulating JHDL circuit objects. With an open API to the circuit structure, application-specific tools can be created to manipulate or interact with JHDL circuit objects.

2.1   Schematic Viewer

Within the design tool is a schematic viewer for displaying the structure of the JHDL circuit. This viewer allows the designer to quickly view the structure and hierarchy of a circuit described by the Java code. In addition, the designer can view the relative layout of FPGA circuits that include performance enhancing placement attributes. Other tools are available for viewing memory contents, signal waveforms, and circuit hierarchy.

2.2 Circuit Netlister

To interface with other tools and design flows, JHDL provides an API for converting a JHDL circuit object into a user-defined data interchange format. Through this API, the structure, interconnect, hierarchy and properties of a circuit described in JHDL is exposed and can be regenerated in one of many possible formats. JHDL currently creates EDIF and VHDL netlists and effort is being made to support other netlist formats such as Verilog. If necessary, user-defined textual or binary interchange formats can be created by exploiting this API.

2.3 Circuit Simulator

An important component of the JHDL design suite is a built-in circuit simulator. This simulator allows designers to quickly test their JHDL circuit within an interactive simulation environment running in the Java Virtual Machine. Simulation models for several FPGA technologies are supported and behavioral models of non-FPGA circuitry can easily be created within Java class objects. This simulator provides an open API for interacting with the simulation environment in an application-specific manner. User defined viewers, functions, testbenchs, or third-party simulators can be linked to the JHDL simulator through this API.

JHDL has been used by several organizations to generate large, high-performance FPGA designs. These designs range in complexity from small, single-chip controllers to large, multi-FPGA designs for automatic target recognition and sonar beamforming. Most designs target FPGA co-processor boards that provide multiple FPGAs, memory, programmable interconnect and a high-speed host interface. The ability to describe the structure of an FPGA circuit in a general-purpose programming language and the set of design aids provided by JHDL has aided the development of many general-purpose FPGA designs and reconfigurable computing applications.

3. JHDL FOR IP DELIVERY

Although JHDL has been used effectively as a design tool, it is especially useful for creating parameterizable module generators. The programming constructs and data structures within Java allow module generator executables to create complex circuit structures based on user parameters and application-specific optimizations. Most module generators exploit technology-specific mapping optimizations and use relative placement to improve timing. Many FPGA module generators have been created in JHDL including a variety of arithmetic, signal processing, logic, and memory modules.

An instance of a JHDL module generator can be created by calling the Java class constructor of the module with the appropriate parameters and options. Once the circuit has been created within JHDL, the user may export the circuit by generating a netlist in one of the supported interchange formats (i.e. EDIF, VHDL, or Verilog). This netlist can then be imported into a users conventional design environment. The following example demonstrates how an instance-specific module is created within JHDL.

3.1 Constant Coefficient Module Generator

A module generator that builds optimized constant coefficient multipliers was created for the Xilinx Virtex FPGA[9]. This module generator creates optimized, preplaced con-

stant coefficient multipliers using partial-product look-up tables. To minimize the area and latency of this circuit, the generated circuit is customized to the specific constant, signal widths, and parameters specified by the user.

An application-specific instance of this constant coefficient multiplier is created by *constructing* the module generator object within Java. The class constructor is called and given all of the appropriate user-specific parameters needed to create the optimized, user-specific instance of the IP. The constructor for this multiplier is shown below:

```
public VirtexKCMMultiplier(Node parent,
                           Wire multiplicand,
                           Wire product,
                           boolean signed_mode,
                           boolean pipelined_mode,
                           int constant);
```

To construct this multiplier, a user would first create `Wire` objects for the multiplicand input and product output signals. The size of these wires is specified by the user when the wires are created. The resulting circuit structure and size is determine by the widths of these signal inputs. For example, if the user specifies an 8-bit multiplicand input, 8-bit constant multiplier, and a 12-bit product, the module generator will create an optimized 8x8 multiplier that provides only the top 12-bits of the product.

Once the wires are created, the user creates variables to represent the various parameters needed by the module generator. In this case, `boolean` variables are used to indicate whether the multiplier is signed and pipelined and an `int` variable is used to represent the actual constant used by the constant multiplier. Once the `Wire` objects and parameter variables are available, the circuit object can be created by instancing the module generator with the Java `new` operator. The following code fragment demonstrates the construction of an 8x8 constant multiplier with 12-bit output using the constant -56.

```
Wire m = new Wire(this,8);  //8-bit input
Wire p = new Wire(this,12); //12-bit output
boolean signed = true;
boolean pipelined = true;
int c = -56;                //constant

new VirtexKCMMultiplier(this,m,p,signed,
                        pipelined, c);
```

After the constant coefficient multiplier has been created, standard JHDL tools may be used to manipulate, view, simulate, or output the circuit into an interchangeable format as needed by the designer.

3.2 Module Generator Executables

While most users of JHDL directly instance module generators within a top-level JHDL design, custom executable programs can be written to deliver a circuit outside of the JHDL design environment. Rather than requiring users to write Java code, an executable can be provided that offers a simple user interface for creating, evaluating, manipulating, and delivering circuits to the user. Although these executables are based on the JHDL design infrastructure, the details of Java and JHDL are hidden from the user.

For example, a simple graphical user interface for delivering the constant coefficient multiplier circuit is shown in

Figure 1. This executable program allows a user to create an application-specific instance of the multiplier by selecting the appropriate module parameters (i.e. bitwidths, constant value, etc.). After selecting parameters, the user may evaluate the multiplier by obtaining area and timing estimates or even simulating the design.
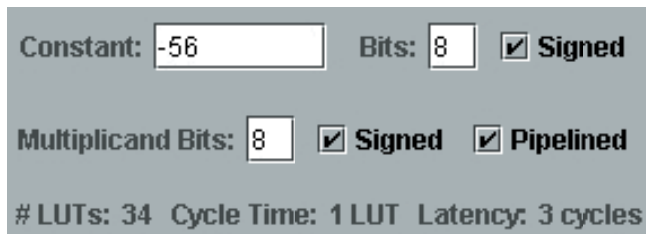

Figure 1: Graphical User Interface for Constant Co-efficient Multiplier.

This executable is relatively straightforward – it contains the JHDL code for generating a multiplier circuit, the FPGA technology libraries, and a circuit estimator. Other more customized executables can be provided that offer different levels of IP visibility. If more visibility is necessary, a structural viewer and netlist generator can be added. If less visibility is desired, the vendor can remove the simulation capability of the executable.

Using an executable to deliver IP allows the vendor to control the content, functionality, and opacity of the IP on an individual basis. A custom Java executable can be created and delivered that is customized to the needs of both the customer and vendor. By controlling the content and opacity of the IP executable, vendors may determine the features available for evaluation as well as the visibility into the delivered IP. Specifically, the IP executable may include any of the following JHDL tools:

Structural circuit viewer The circuit viewer allows to user to browse the hierarchy and structure of a generated design. A structural circuit viewer may be included within the executable to provide the user the ability to examine the circuit structure and interface (see Figure 3).

Executable simulation model The JHDL simulator may be included within the executable to allow simulation of the generated circuit. If necessary, the self-contained functional simulator can be used in conjunction with a user's own simulation tools to evaluate the functionality of the IP within a larger design.

Programatic circuit generator interface Most reusable circuit modules for FPGAs are parameterizable and require user intervention to determine the bit-widths, style, pipelining and other parameters of the circuit in question. IP executables may provide an interface that exposes the parameters and options available to the user of the IP. Using this interface, customers may experiment with various parameters to estimate the speed, size and cost of the IP.

Layout view For many FPGA circuit modules, the relative placement or layout of the module has a significant impact on the overall timing and placement of the final FPGA design. A view of the layout for pre-placed FPGA macros provides the user with feedback on the size, shape, and layout of a circuit module under review. Users may explore various placement and layout options of a macro without seeing the underlying circuit structure or netlist.

Circuit netlisting The executable must provide a way for delivering the IP in a form acceptable by the user's conventional tool chain. For licensed users, the circuit netlisting libraries can be included to generate netlists of the IP in the appropriate netlist format. Structural VHDL, Verilog, or EDIF may be generated with the appropriate technology specific mapping constraints for the circuit module of interest.

These and other application-specific IP evaluation and delivery tools may be organized into a single executable on a customer by customer basis. Two such executable configurations are depicted in Figure 2. For passive customers that wish to browse a given circuit and its corresponding characteristics, limited visibility into the IP is provided. As suggested by the left configuration, an executable can be created that contains an interface to the IP and a basic circuit estimator. For more active or licensed customers, an executable can be created that provides greater visibility. As shown by the configuration on the right, the IP delivery executable may contain circuit layout viewers, high-level structural views, and a simulation model. A wide variety of such IP delivery executables may be created by combining the various JHDL design aids.
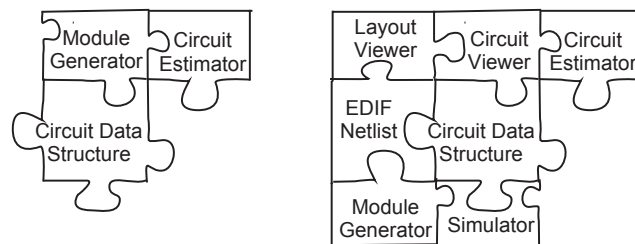

Figure 2: Two configurations of an IP delivery executable.

4. IP DELIVERY APPLETS

A simple way of delivering such JHDL circuit generator executables to a customer is through the internet in the form of Java applets. Rather than delivering circuit generators directly to the customer via mail or email, customers can directly access and execute these generator applets within their standard web browser. Delivering circuit generators through the internet offers a number of advantages over traditional methods: users do not need to install software on their machine, the IP delivery applets can be maintained at a central server, and access to such applets can be controlled through a variety of web-based security measures. By accessing applets over the internet, users are able to evaluate, test, and obtain intellectual property from an IP provider directly from a web browser.

There are a variety of ways in which the IP delivery concepts using JHDL applets can be used. This section will

outline two specific scenarios in which the JHDL infrastructure has been used to test the IP-delivery capability within our laboratory. Sample JHDL IP delivery applets can be found and tested online at the following web URL: *http://www.jhdl.org/applets.*

4.1 Constant Coefficient Multiplier Applet

Figure 3 demonstrates a simple applet used for evaluating and delivering the constant coefficient multiplier described earlier. This applet uses a JHDL module generator to create an optimized constant multiplier based on the user's application-specific needs. Rather than exposing the software API to the user, this applet provides a GUI interface to the user for specifying the circuit parameters and evaluating the suitability and quality of the resulting design.
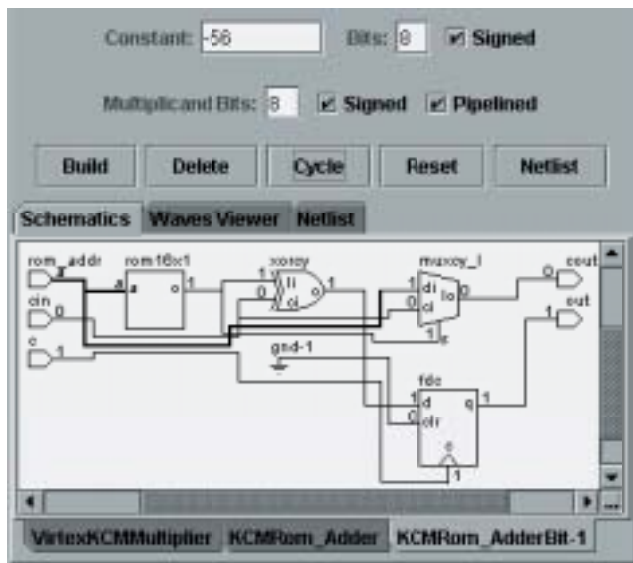

Figure 3: Sample Applet for Constant Coefficient Multiplier.

A user evaluates a specific instance of the IP by selecting the desired parameters and pressing the "build" button. When this event occurs, the applet creates the constant multiplier by calling the module generator constructor. At this point, the circuit is created within the JHDL data structures and any of the JHDL tools can be used to manipulate, netlist and view the circuit structure. In the applet example of Figure 3, a schematic viewer is drawn and a user can interactively explore the structure and hierarchy of the created circuit.

If the JHDL simulator is included with an applet, the user is able to simulate the behavior of the generated circuit directly within the web browser. Using the **Cycle** and **Reset** buttons, a user may cycle the simulator and browse the state of the circuit throughout the circuit hierarchy. The history of the circuit state can be recorded and viewed using the JHDL waveform viewer.

Once a circuit module and its application-specific parameters are selected by a user, the applet may provide a way of delivering the circuit to the user in a reusable form. Since a data structure representing the circuit is resident within the applet, any one of the JHDL circuit netlisting tools can be

used to generate a reusable representation of the circuit. For this specific applet, a **Netlist** button is given that generates an EDIF netlist of the application-specific circuit. The generated netlist is displayed in a scrollable text window for the user to browse or copy.

4.2 Black Box Simulation Applet

The previous applet example demonstrates a relatively transparent IP delivery applet – the users of such applet can browse the hierarchy of a circuit, simulate the internals of the circuit, and generate a reusable netlist from the instance-specific circuit. In many cases, the applet executables will need to limit access to the internals of the generated circuits.

One approach for limiting internal access is to provide a "black box" simulation model of the generated circuit. Like the previous "transparent" example, a user would interact with the applet to choose instance-specific circuit parameters and options to build a circuit. Unlike the previous example, the user does not have the ability to browse the hierarchy of the circuit or obtain a netlist. Instead, the applet includes a self-contained simulation model of the intellectual property. This simulation model may be operated interactively by the user or integrated into the user's conventional design flow.

Figure 4 demonstrates a black box simulation model of two IP delivery applets operating within a user's web browser. In this scenario, the user wishes to simulate two external circuits with other components in a complete system simulation. To create this simulation environment, the user visits the web page associated with each IP and constructs the IP within the applet. Next, the user establishes a connection between each applet and the system simulator using an application-specific communication protocol such as RMI or socket communication[1]. Once an appropriate communication link established between these simulators, the entire system can be simulated together without exposing the internals of the applet-based IP.
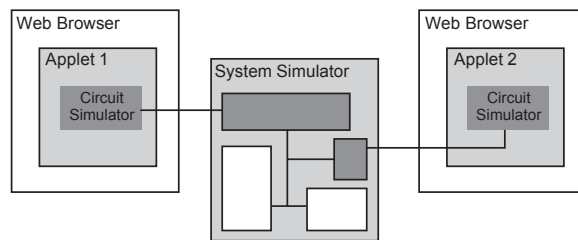

Figure 4: Integrating Black Box Simulation Model Into Simulation Environment.

Several JHDL designs were tested using this black-box simulation methodology. A simulation wrapper was created to interface the JHDL black-box simulator with a Verilog simulation using PLI[8]. Simulation events are exchanged over network sockets and a custom communication protocol. By exposing the JHDL black-box simulator, a user can evaluate intellectual property within their design environment without exposing any proprietary information.

---

[1]Establishing network connections between applets operating on a client browser and other client applications violates the default applet security model and requires explicit permission from the user.

This approach of black-box simulation is similar to the client-server simulation architectures proposed by [1] and [2]. In these architectures, IP can be simulated on a customer client machine through a pre-arranged communication protocol with a vendor supported server. The IP is protected by keeping the simulation models at the vendor site and only providing the appropriate simulation state. Unlike these methods, the IP in this approach is simulated on the *client* machine. This offers the the potential for IP simulation with much lower network latency than possible with off-site simulation models.

4.3 IP Applet Security

An important concern associated with IP delivery using Java Applets is preserving the security of the intellectual property contained within the applet. Since the IP applet executable is accessible by the user, a sophisticated user may attempt to uncover the IP by reverse engineering the applet. IP applets may need to provide additional measures of security to properly protect the IP. Techniques such as Java class file obfuscation and class encryption may be added to increase the security of the IP. Other more standard approaches for IP protection may be used such as watermarking[7] and hardware metering[6].

4.4 Applet Download Time

An important issue that faces users of Java applets is the size of the program binary downloaded by the browser. Since the binaries are loaded by the browser the first time the web page is accessed, large binaries may require an unreasonable amount of time and network bandwidth. To address this issue, the binaries associated with the JHDL design tool are partitioned into a number of smaller, more specific Jar archive files[2]. This allows a given applet to require only those Jar files required by the applet code and avoid wasting network bandwidth with unused class files. As shown in Table 1, the executables required by the applet of Figure 3 involve four such jar files for a total of 795 kB. Other jar files may be provided for applets requiring additional functionality or technology libraries.

File	Size	Description
JHDLBase.jar	346 kB	JHDL Classes & Simulator
Virtex.jar	293 kB	Xilinx Virtex Library
Viewer.jar	140 kB	Schematic Viewers
Applet.jar	16 kB	Module Generator & Applet
Total	795 kB	

Table 1: JAR Files Used By Constant Multiplier Applet

5. CONCLUSIONS

Exploiting the functionality of the JHDL design environment within Java applets provides a powerful and flexible approach for web-based IP evaluation and delivery. By using the JHDL infrastructure, applets can be created with a variety of IP evaluation and delivery features. Schematic viewers, simulators, netlist generators and other related functionality can be combined into an applet that provides appropriate levels of visibility along with IP protection. Since these applets run on a browser and are accessed from the web, IP evaluation is made available to customers with little or no external software. Further, customers will always access the latest revisions of the IP applet software when the applet binaries are downloaded.

Future efforts to improve JHDL applets include creating applets for more complicated IP, investigating more secure delivery techniques, and developing applets that delivering more than one IP module. As additional IP evaluation features are developed and interfaces with more tools are provided, JHDL based applets will provide a powerful alternative for web-based delivery of FPGA IP.

6. REFERENCES

[1] M. Dalpasso, A. Bogliolo, and L. Benini. Virtual simulation of distributed ip-based design. In *Proceedings of the 36th Design Automation Conference, DAC 1999*, pages 50–55, 1999.

[2] A. Fin and F. Fummi. A Web-CAD methodology for IP-core anaylsis and simulation. In *Proceedings of the 37th Design Automation Conference, DAC 2000*, pages 597–600, 2000.

[3] B. Hutchings, P. Bellows, J. Hawkins, S. Hemmert, B. Nelson, and M. Rytting. A CAD suite for high-performance FPGA design. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 12–24. IEEE Computer Society, IEEE, April 1999.

[4] B. L. Hutchings and B. E. Nelson. Using general-purpose programming languages for FPGA design. In *37rd Design Automation Conference (DAC)*, pages 561–566, Los Angeles, CA, June 2000.

[5] P. James-Roxby and S. A. Guccione. Automated extraction of run-time parameterisable cores from programmable device configurations. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 153–161, April 2000.

[6] F. Koushanfar and G. Qu. Hardware metering. In *Proceedings of the 38th Design Automation Conference, DAC 2001*, pages 490–493, 2001.

[7] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Robust FPGA intellectual property protection through multiple small watermarks. In *Proceedings of the 36th Design Automation Conference, DAC 1999*, pages 831–836, 1999.

[8] S. Mittra. *Principles of Verilog PLI*. Kluwer Academic Publishers, 1999.

[9] M. J. Wirthlin and B. McMurtrey. Efficient constant coefficient multiplication using advanced FPGA architectures. In *Field-Programmable Logic and Applications. Proceedings of the 11th International Workshop, FPL 2001*, pages 555–564, August 2001.

---

[2]Java Jar files are compressed archive files used to collect a number binary class files and other program resources.