BoardScope: A Debug Tool for Reconfigurable Systems

Delon Levi and Steven A. Guccione

Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124

ABSTRACT

BoardScope is a portable, interactive debug tool for Xilinx FPGA-based hardware. *BoardScope* features simple but powerful graphical interfaces for viewing FPGA circuits in their operational state. The main display graphically shows the *Configurable Logic Block* (CLB) flip-flop states of all FPGA devices in the system. This display indicates overall dataflow and is used to find design errors, such as clocking problems or incorrect initialization or reset. The CLB display shows the complete state, including LUT values and flip-flop configuration, of any CLB. Finally, a symbol file may be used to drive a waveform display. The waveform display permits both bit-level signals and multibit busses to be viewed in a fashion similar to that used by circuit simulators. *BoardScope* is implemented completely in Java, using the *Abstract Window Toolkit* (AWT) version 1.1. The interface to the hardware is provided by *XHWIF*, the *Xilinx standard HardWare InterFace* for FPGA based hardware. This simplifies porting, and provides remote access capabilities. Remote access allows multiple users to communicate with the hardware across a network. BoardScope currently runs on the *WildForce*™ and *WildOne*™ boards from *Annapolis Microsystems*, and the *PCI Pamette* from *Digital Equipment Corporation*. Ports to other systems are currently under way.

Keywords: FPGA, debug, reconfiguration, Java

1. INTRODUCTION

Various hardware platforms have been built [7], but software support for these systems has lagged. In the area of software support, the emphasis has been almost exclusively on design tools. By contrast, little has been reported on debug environments for these systems. Among the debug tools mentioned in the literature are *the systolic parallel C (spC)* deubgger for the *Enable++* system [9], the *KRONO* and *ShowRB* debugger for the *PAM* system [5], the *T2* debugger for the *Splash 2* system [4], the *Hardware Promela Debugger (HPDB)* for the *PROMELA* system [10], and *DISC debugger (DDB)* for the *DISC* system [6] and the *CALLAS* debugger for *CHAMELON* [8]. While this is a substantial number of systems, none are designed to support new boards.

BoardScope is a tool for graphically examining the operation of FPGA circuits on any reconfigurable computing board. Like circuit simulators, it is used to verify the design's operation. Like in-circuit emulators, the results are produced by the operations of actual hardware rather than from software models. This capability provides a more accurate verification, and furthermore, it enables debugging FPGA circuits while they are communicating with other hardware components.

BoardScope developed from work originally done on the *WebScope* debugger. *WebScope* specifically targeted the Xilinx XC6200™ devices and the board interface was designed specifically for the *Xilinx XC6200DS Development System*. In addition, *WebScope* only communicated with a single FPGA. *BoardScope* on the other hand, has been upgraded to support debugging of designs operating on multiple FPGAs at the same time. Also, using the Xilinx HardWare InterFace, *BoardScope* can communicate with a variety of FPGA-based boards.

While *WebScope* used the XC6200's open hardware interface to directly access the FPGA's internal resources, *BoardScope* uses the *Xilinx Bitstream Interface* (*XBI*) to access resources in the FPGA's bitstream. Then using *XHWIF*, the bitstreams are downloaded to configure the FPGAs, or readback to analyze them (see Figure 1). Currently, *BoardScope* supports devices in the Xilinx XC4000EX™ and the XC4000XL™ families.

Further Author Information
D.L E-Mail: Delon.Levi@xilinx.com
S.A.G. E-Mail: Steven.Guccione@xilinx.com

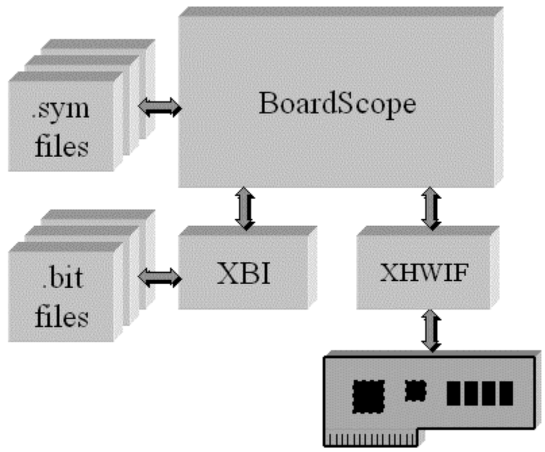

Figure 1 - The software structure

2. THE MAIN DISPLAY

The FPGA Watch graphically displays the states of all CLB flip-flops for all computational FPGAs on the board. Figure 2 shows the FPGA Watch for a *Pamette* board with four Xilinx XC4028EX FPGAs. Each major squarerepresents the CLB array for a single FPGA. The coloration of the smaller tiles in each square indicates the state of the X or Y CLB flip flop: blue denotes a high state, while green denotes a low state. Clicking on a particular FPGA zooms the view to that FPGA. If the states of the FPGAs are changed, either through a reset or by incrementing the on-board clock, the tiles are repainted to indicate the new states. The display therefore enables examination of all CLB flip flops in one view, and further enables examination of the flips flops as they change from one state to the next. The FPGAs in Figure 1 are loaded with a linear cellular automata demonstration circuit. The unique triangle pattern prdouced by such automata is visible on the display.

Because the display maintains the spatial arrangement of the resources on the silicon, the states of physically arranged macros, like column counters, can be conveniently monitored. For example, bits in a column shift register can be seen shifting up, as the clock is single-stepped. Furthermore, if the macros are connected in an ordered datapath, the states of the macros can be examined as data propagates through the pipeline. The global view also enables quick determination of missing clocks, missing resets, or double clocks. So the FPGA Watch is optimal for detecting problems in physically structured designs, although for unstructured designs, the display can be useful in detecting problems that affect large portions of the circuit.

For a more detailed view of the circuit, the state of a Configurable Logic Block can be seen in the CLB display (see Figure 2). The display shows the F, G, and H look up table states, the X and Y flip-flop configuration and states, and the CLB's internal interconnect. Clicking on a tile in the FPGA Watch Display updates the CLB Display for the CLB. Because this is a detailed view, it is probably most useful to those intimately familiar with the circuit implementation and XC4000EX/XL™ CLB architecture.

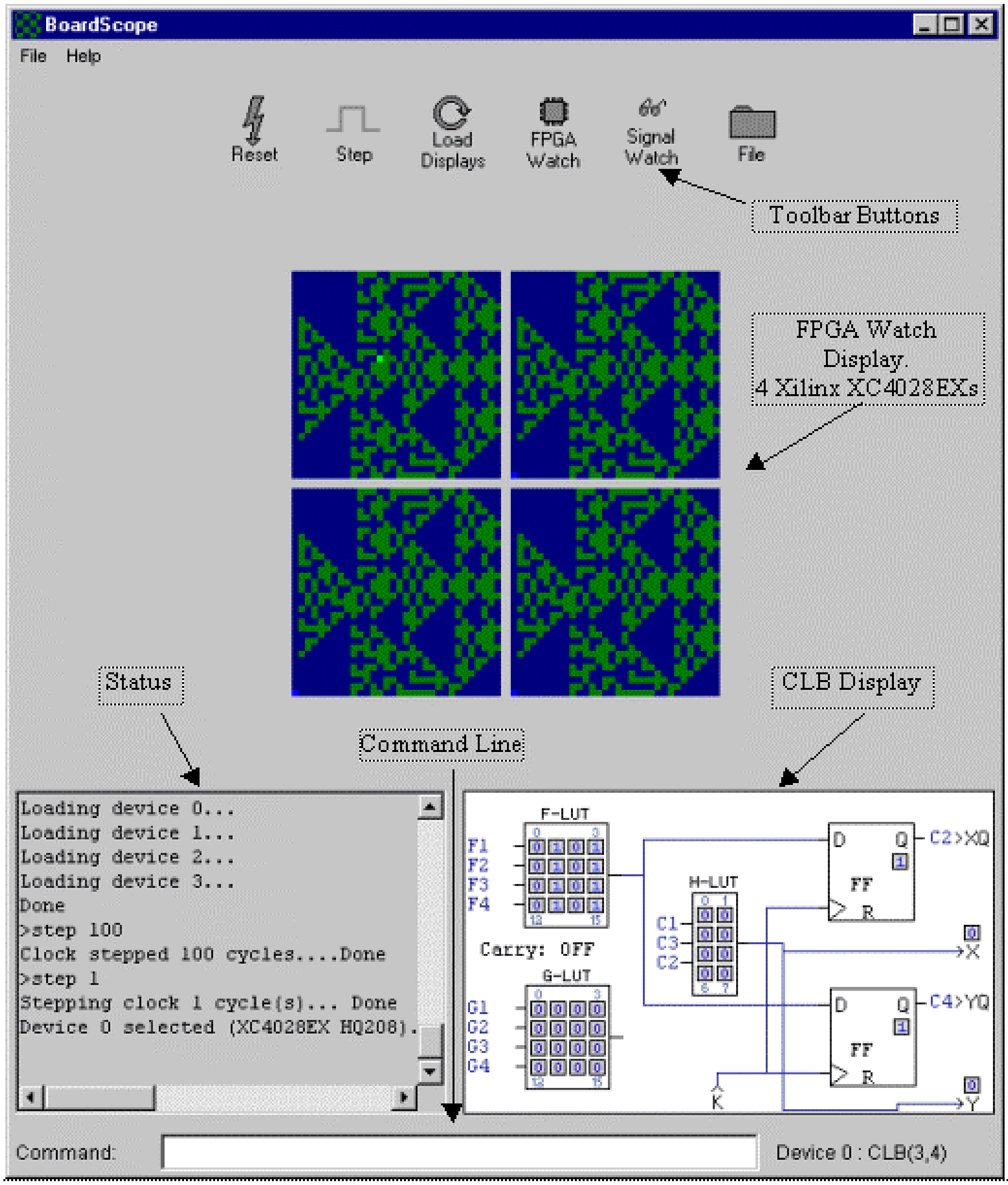

Figure 2 - BoardScope Main Display

3. THE WAVEFORM DISPLAY

A user can select specific points to probe in the circuit, and view them using the Signal Watch waveform display (see Figure 3). The states of the signals are displayed both numerically, and graphically as waveforms. As the clock is stepped, the waveforms are updated to reflect the current state. Single probe signals are drawn as square waves, whereas multi-probe signals (which are used to examine busses) are drawn with the vertical axis representing the signal value. If multiple clock steps are executed at once, for example through a step 100 command, then an ellipsis (…) is shown to indicate the unknown intermediary states, followed by a waveform to indicate the final state. The numerical fields, located by the signal names, reflect the states of the signals at the cursor. Clicking the cursor at a past state updates the numerical fields.

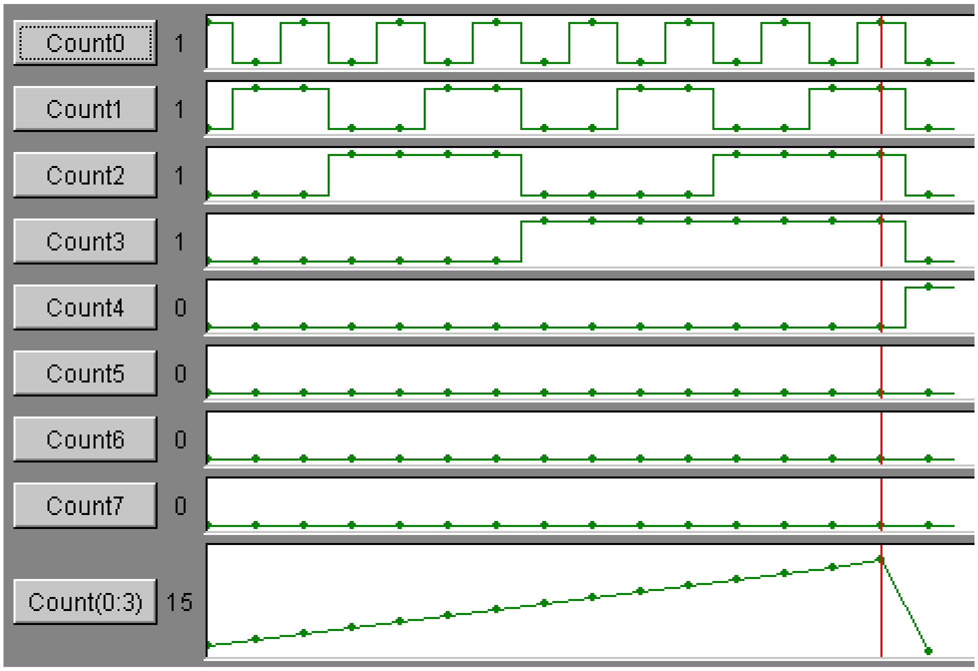

Figure 3 - Signal Watch Waveform Display

To specify signals, the user creates an ASCII .sym file like the one shown in Figure 4. For each signal, a name and a list of one or more probe points are specified. Valid probe points are CLB XQ and YQ flip-flops/latches, or CLB X and Y outputs. These points have latches in the FPGA, which enables their states to be captured in the readback bitstream and then graphically displayed. To initialize the Signal Watch display with new signals, a .sym file is selected using the *BoardScope* file dialog.

To probe signals in designs created using the *Xilinx M1* or *XACT* tools, a utility is included which translates a .ll file into a .sym file. The .ll file contains single bit probe points for the associated .bit file. For probes that are part of a bus, the translator automatically aggregates the probes into busses, based on their names and indices, and writes them to the .sym file.

```
# Sample Entries:
#
# Name = {deviceNumber:source, ...deviceNumber:source, deviceNumber:source}
#
# sum[0:3] = {clb(5,4).xq clb(6,4).yq clb(7,4).xq}
#
# select = {clb(2,15).xq}
Count0 = {clb(10,10).xq}
Count1 = {clb(11,10).xq}
Count2 = {clb(12,10).xq}
Count3 = {clb(13,10).xq}
Count4 = {clb(14,10).xq}
Count5 = {clb(15,10).xq}
Count6 = {clb(16,10).xq}
Count7 = {clb(17,10).xq}
Count(0:3) = {clb(10,10).xq
              clb(11,10).xq
              clb(12,10).xq
              clb(13,10).xq}
```

Figure 4 - The .sym file format

4. THE USER INPUT INTERFACE

The primary control provided by *BoardScope* is via the toolbar buttons across the top of the window. These buttons control the states of the display and the states of the hardware.

The **Reset** button clears the configuration memories of all of the FPGAs, so that they are set to their initial power-on state. This control should not be confused with resets that initialize the design flip-flops to their initial state; this reset actually removes the circuits from the FPGAs. If a circuit with an illegal configuration - is loaded in the FPGAs, for example one with multiple outputs driving the same line, the **Reset** button can be used as a panic button to remove the circuit before serious damage occurs.

The **Step** button increments the state of the board. Pressing the button instructs the oscillator on the board to send a single clock pulse, which is received by the FPGAs and updates the flip-flops.

The **Load Display** button updates the main FPGA flip-flop display and the Signal Watch display. This is particularly useful for situations when the board is already in an operational state when *BoardScope* is started, perhaps through another software package. *BoardScope* is then used to read and increment the states without re-initializing the FPGAs.

The **FPGA Watch** button is used to show all of FPGA flip-flops in the Main Display. This button is useful for situations when the Main Display is zoomed onto a single FPGA, or when it displays the Signal Watch waveforms.

The **Signal Watch** button switches the Main Display to the Signal Watch waveform display. This button is not enabled until a .sym file is loaded using the **File** button.

The **File** button enables a .bit or .sym file to be loaded. Loading a .bit file programs the FPGAs with the circuit described in the file. If one FPGA is showing in the Main Display, only that FPGA is programmed with the circuit, otherwise all of the FPGAs are programmed with the circuit. Loading a .sym file creates a Signal Watch display with signals that are defined in the .sym file.

A secondary control for *BoardScope* is available through the command line, located at the bottom of the window. This interface provides complete control over the hardware. On-line help is also available from the command line. This gives more detailed descriptions, through the text status area above the command line, of the available commands and their syntax.

5. SAMPLE FLOW

When *BoardScope* is first brought up, it shows a checkered FPGA Watch indicating the display's initial state. Using the **File** button, a .bit file is selected, which configures all of the FPGAs, and updates the display with the FPGAs' initial state. Using the **File** button again, a .sym file is selected, which sets up a Signal Watch waveform display. A `step 1000` command is given on the command line to quickly increment the hardware to a known state. The **Step** button is clicked several times, which updates the waveform display with the signal states. The FPGA Watch button is clicked, revealing that several macros in upper left corner are still in their initial state. The bug is found and the **Reset** button is clicked to clear the FPGAs. *BoardScope* is closed using the `File->Quit` menu.

6. THE XHWIF INTERFACE

XHWIF, the Xilinx hardware interface standard, is a Java interface for communicating with FPGA-based boards. It includes methods for reading and writing bitstreams to FPGAs, and methods for describing the kinds and number of FPGAs on the board. Also included are methods for incrementing the on-board clock, and for reading and writing to on-board memories, if they are available. Essentially, the interface describes the board, and enables sending data on and off the board.

The interface standardizes the way that applications communicate with hardware, so that using the same interface, applications, like *BoardScope* for example, can communicate with a variety of boards. All of the hardware specific information is hidden inside of a class that implements the XHWIF interface. Using the *Java Native Interface (JNI)*, which allows Java programs to interface with C programs, calls to the interface are converted into calls to the board's drivers. This methodology frees BoardScope, or other applications, from communicating directly with the drivers or the bus. In fact, hiding the bus and driver specific information in the class implementing the XHWIF interface enables applications to communicate with boards connected through any bus or communications link, whether it be PCI, ISA, or some standard yet to be defined. The driver performs all the bus interactions. To port a new board to BoardScope, only a Java class that implements the *XHWIF* interface needs to be created.

7. THE XHWIF SERVER

The *XHWIF* server is an application that implements the *XHWIF* interface (see Figure 5). It enables other applications, like BoardScope, to communicate with reconfigurable computing boards located anywhere across the Internet. A board can be installed thousands of miles away, and the server still allows applications to communicate with it. This capability enables design debug without having direct access to the hardware, and it further enables multiple users to access the board.

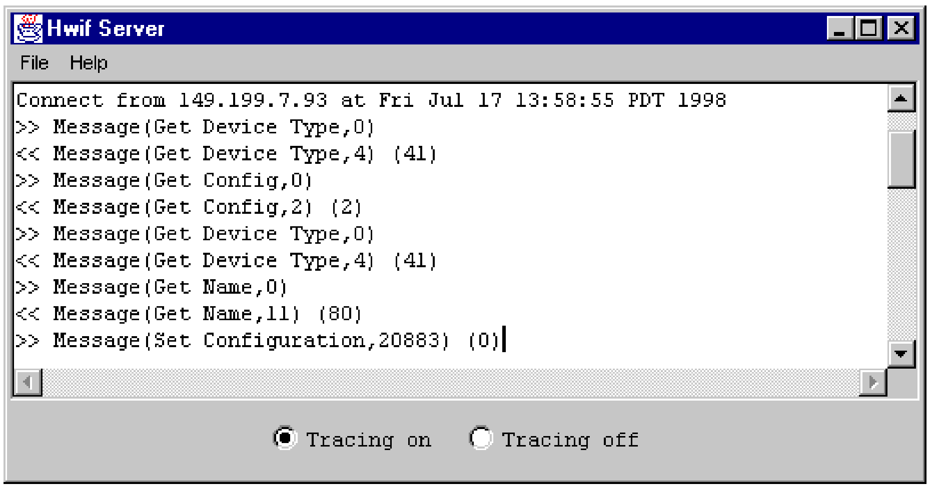

Figure 5 - XHWIF Server

BoardScope's functionality is exactly the same whether it is using the server or talking with a locally installed board. The only difference is that the FPGA configuration and readback times are slower. This arises from the Internet's limited bandwidth.

Figure 5 shows the server. If tracing is turned on, detailed statuses of the message passing are displayed. To establish the communication link, the server is started on a machine that has a board installed locally, and *BoardScope* is launched on a remote machine using the following flag:

-HWIFNet <machine name>

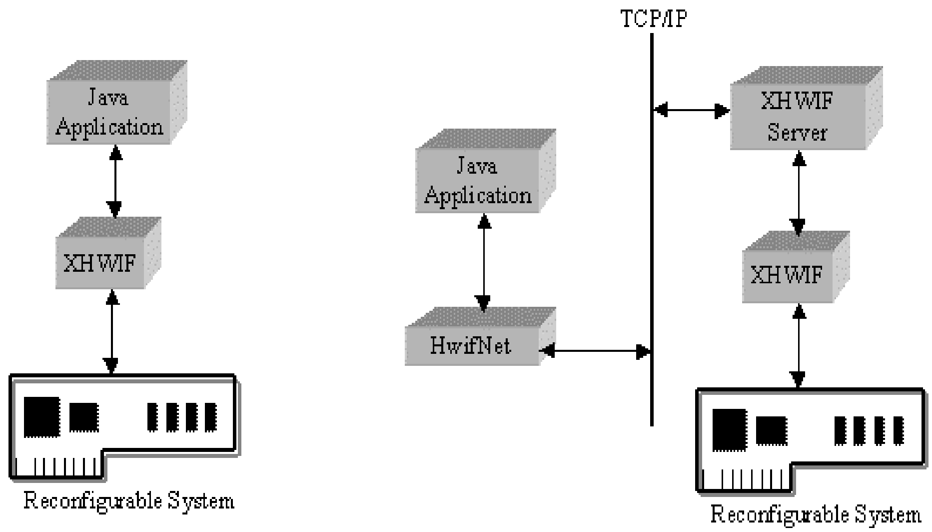

Figure 6 - Local and remote access to the target hardware

Other Java-based applications can also use the *XHWIF* server to remotely communicate with reconfigurable computing boards. The server comes with a HWIFNet Java class that plugs into the user interface. It implements the XHWIF interface exactly like other board interface classes that communicate directly with the hardware. Therefore, the code overhead of communicating with the server is only a few lines to determine whether the server connection is requested, and then to declare the board interface object as a HWIFNet type. No other board specific distinctions need to be made. For example, in the code below, if the server connection is selected, then the board interface object is created as a HWIFNet type, otherwise the board interface object is created WildOne type[1]. The rest of the code for the board interface object, for example the getConfiguration() and setConfiguration() methods, are exactly the same.

```
if (server == true)
        Hwif board = new HWIFNet();
else
        Hwif board = new WildOne();
```

Figure 7 - Sample Code for integrating the XHWIF server

8. CONCLUSIONS AND FUTURE WORK

BoardScope is fast interactive verification aid. It provides symbolic debug for high level applications, and physical-level debug for detecting low level problems. *BoardScope* is particularly useful for developers creating tools for reconfigurable systems. Unlike software verification tools, it enables monitoring the effects of reconfiguration on the design, and furthermore, it generates data from the actions of real hardware. Unlike hardware-based verification tools, like boundary scan or logic analyzers, which only monitor IOs, *BoardScope* can monitor the states of internal logic. without having to modify the design by routing probe points to the IOs.

BoardScope can also target traditional ASIC type uses of FPGAs, where the logical structure remains static over the design's operational life. In these situations, *BoardScope* can be used to determine the internal structure and states of the design, which can be useful in discovering discrepancies between simulation, and the actual hardware implementation and execution.

The *XHWIF* hardware interface and the associated HWIF server are broken off as stand-alone modules that enable a variety of applications, *BoardScope* among them, to communicate with a variety of reconfigurable systems. In addition, *XHWIF*

[1] The WildOne object is a board interface for a locally installed *Annapolis WildOne*™ board.

eases development by standardizing the protocol for communicating with the hardware. Finally, *XHWIF* enables shared and remote access to the hardware.

Because of the object oriented nature of Java, which allows new modules to be easily brought in, much room for improvement exists. *BoardScope* can be ported to new architectures, like the Xilinx Virtex™ series. In addition, the physical displays can be updated to show a more comprehensive view of the intra-CLB routing. For example, when clicking on a CLB, the CLB's fan-in and fan-out can be shown as in-coming and out-going arrows. This particular functionality would be similar to that found in the Xilinx *XACT Floorplanner*. Alternatively, the general intra-CLB routing can be graphically displayed as part of the CLB array structure, in a manner similar to the Xilinx *XACT EditLCA* and *M1 EPIC* displays. For partially configurable architectures in particular, this feature would allow monitoring the changes in a design's interconnect after each configuration.

9. REFERENCES

1. S. A. Guccione, "WebScope: A circuit debug tool", *Field Programmable Logic 98*, Accepted for Publication, Tallin, Estonia, September 1998.
2. S. A. Guccione and D. Levi, "Xilinx Bitstream Inteface: A Java-based interface to FPGA hardware", in *Confiurable Computing Technology and its uses in High Performance Computing, DSP and Systems Engineering, Proc. SPIE Photonics East,* J. Schewel, ed., SPIE - THE International Society for Optical Engineering, (Bellingham, WA), November 1998.
3. Xilinx, Inc., The Programmable Logic Data Book, 1996.
4. J. M. Arnold, "The Splash 2 Software Environment", in *IEEE Workshop on FPGAs for Custom Computing Machines*, D. A. Buell and K. L. Pocek, ed., pages 88-101, IEEE Computer Society Press, Los Alamitos, Ca, April 1993.
5. P. Bertin and H.' Toutai., "PAM programmming environments: Practive and Experience", in *IEEE Workshop on FPGAs for Custom Computing Machines*, D. A. Buell and K. L. Pocek, ed., pages 133-108, IEEE Computer Society Press, Los Alamitos, Ca, April 1993.
6. D. A. Clark and B. L. Hutchings, "Supporting FPGA microprocessors through retargetable software tools", in *IEEE Workshop on FPGAs for Custom Computing Machines*, D. A. Buell and K. L. Pocek, ed., pages 195-203, IEEE Computer Society Press, Los Alamitos, Ca, April 1993.
7. S. A. Guccione, "List of FPGA-based computing machines", World Wide Web page http://www.io.com/~guccione/HW_list.html, 1997.
8. B. Heeb and C. Pfister, "Chameleon: A workstation of a different colour", in *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping, Proceedings of the 2nd Interanational Workshop on Field-Programmable Logic and Applications,* H. Grunbacher and R. W. Hartensteing, ed., FPL 98, Lecture Notes in Computer Science 705, pages 152-161.
9. H. Hogl, A. Kugel, J. Ludvig, R. Manner, K. H. Noffz, and R. Zoz, "Enable+++: A second generation FPGA processor", *IEEE Symposium on FPGAs for Custom Computing Machines*, P. Athanas and K. L. Pocek, ed., pages 45-53, IEEE Computer Society Press, Los Alamitos, Ca, April 1995.
10. A. Wenban and G. Brown, "A software development system for FPGA-based data acquisition systems", *IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, ed., pages 28-37, IEEE Computer Society Press, Los Alamitos, Ca, April 1996.