JBits: Java based interface for reconfigurable computing

Steve Guccione, Delon Levi and Prasanna Sundararajan
Xilinx Inc.,
2100 Logic Drive
San Jose, CA 95124 (USA)
steve.guccione@xilinx.com
delon.levi@xilinx.com
prasanna.sundararajan@xilinx.com

The *JBits*™ software is a set of Java™ classes which provide an Application Programming Interface (API) to access the Xilinx FPGA bitstream. The interface operates on either bitstreams generated by Xilinx design tools, or on bitstreams read back from actual hardware. This permits all configurable resources like Look-up tables, routing and the flip-flops in the FPGA to be individually configured under software control.

The API has been used to construct complete circuits and to modify existing circuits. In addition, the object-oriented support in the Java programming language has permitted a small library of parameterizable, object oriented macro circuits or Cores to be implemented. Finally, this API may be used as a base to construct other tools. This includes traditional design tools for performing tasks such as circuit placement and routing, as well as application specific tools to perform more narrowly defined tasks.

The circuits developed can be downloaded on to the Xilinx hardware and probed using *BoardScope. BoardScope* is a graphical and interactive hardware debug tool for Xilinx FPGAs. It enables a user to look inside the chips and see the internal states and circuit configurations while the hardware is operating. The data is sampled using the readback capabilities of the FPGAs, and then graphically displayed. The interface to the hardware is provided by *XHWIF*, the *Xilinx standard HardWare InterFace* for FPGA based hardware.

XHWIF interface permits simple porting of both *JBits* and *BoardScope* to new hardware platforms. Once the XHWIF interface is defined for a particular piece of hardware, tools such as *BoardScope* and *JBits* based Java applications will run without any recompilation or modification. In addition, the *JBits* API and *XHWIF* interface can be suitably integrated to design run -time reconfigurable applications. Finally, part of the *XHWIF* package is TCP/IP based remote network access support. This enables remote hardware configuration and debugging capabilities.

Thus the *JBits* API along with *BoardScope* and *XHWIF* enables run-time reconfigurable application development, hardware debugging, and remote hardware configuration capabilities.

Keywords: FPGA, Reconfiguration, Java, Remote hardware configuration, Hardware debugging

1. INTRODUCTION

The *JBits* Bitstream Interface is a set of Java classes which provide an Application Program Interface (API) into the configuration bitstream for devices in the Xilinx XC4000™ and Virtex™ families. This interface permits all configurable resources in the device to be individually programmed under software control. This provides software support for a set of new capabilities previously unrealized in Xilinx devices.

Using the *JBits* interface, software can be written which produces circuits, and provides support for dynamic reconfiguration of these circuits. In addition, because the entire system is implemented in the Java programming language, any existing Java development environment may be used with *JBits* API. This provides a simple alternative to existing CAD-based tools.

2. THE DESIGN FLOW

The *JBits* API is based on the Java Environment for Reconfigurable Computing for the XC6200 ™ family of devices (JERC6K). JERC6K was also implemented completely in Java and provided fast compile times and supported dynamic reconfiguration. In some sense, *JBits* may be viewed as a version of JERC6K for the XC4000 and Virtex families.

The original motivation for *JBits* was to support dynamic reconfiguration in the Xilinx XC4000 and Virtex family of parts. While dynamic reconfiguration has always been possible in all the Xilinx SRAM-based parts, very little has been done to provide software support for this capability. In general, the design flow has been limited to static circuit design tools, with schematic capture or Hardware Description Language (HDL) front-ends. Clearly with such static design methodologies, supporting reconfiguration would not be possible.

In addition, the method used to produce configuration data from these circuits was based on automatic placement and routing technology developed originally for production of printed circuit boards. This approach relied on the solving of known NP-complete problems and was necessarily slow and non-deterministic. Finally, placement algorithms usually provided a physical implementation of the circuit which bore little resemblance to the logical circuit. This made the task of locating items for reconfiguration difficult.

This led to some simple requirements for a software tool to support dynamic reconfiguration. The tool would have to be as fast as possible, and provide physical information about the circuit for reconfiguration. This all but ruled out traditional CAD approaches.

The solution was to supply a library which gave complete access to all of the configurable architectural features of the device. Because the library would be pre-complied Java classes, the result would not be a static configuration bitstream, but rather executable code. This code would execute and supply configuration control and data to the reconfigurable logic. Figure 1 illustrates the *JBits* design flow.

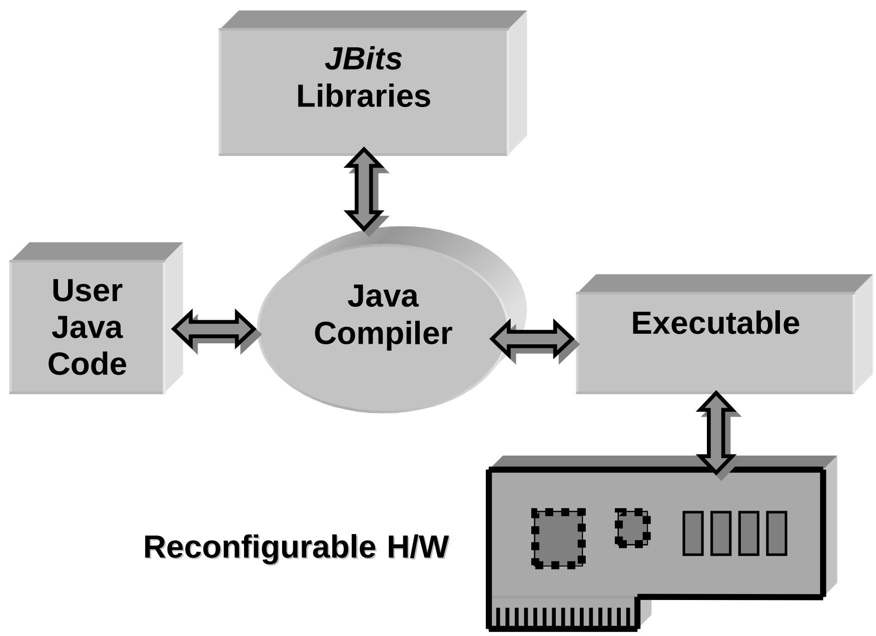

Fig 1. JBits Design Flow

One result of this model is that the executable is just an arbitrary piece of compiled Java code. Not only does this make the resulting software portable across a wide variety of systems, but also it permits a tight integration with other portions of the system. For instance, a complete GUI for the reconfigurable application may be part of this executable. Such integration has many benefits. The largest benefit is that information is easily shared between the host processor and the FPGA. In most other reconfigurable computing systems, the FPGA circuit design portion of the system is completely decoupled from the host software interface. Not only is this error prone, but it makes errors difficult to find and modifications difficult to make. With a single integrated piece of software which does both circuit configuration and host management, maintaining consistency between the host interface software and the FPGA circuit is greatly simplified.

3. THE *JBITS* SYSTEM DESIGN

Figure 2 illustrates the *JBits* system design. At the center is a user-written Java program. This program makes use of the *JBits* interface to manipulate FPGA family configurable resources that include the look-up tables, routing and flip-flops. Each function call at the *JBits* interface level makes one or more calls to the Bit Level Interface. At this level, a single bit in the bitstream is configured or cleared. Manipulating groups of such bits at the *JBits* level supplies a simpler abstraction. While individual bits may be set using the Bit Level Interface, it is not likely that users will ever make direct use of this low-level interface. The Bit Level Interface is provided as a layer which provides the necessary support for all devices in a given family. That is, the Bit Level Interface is responsible for knowing the bit location in the bitstream of a given bit of configuration data for any device in the XC4000 and Virtex family. Without this layer, custom interfaces would have to be generated for each device in a family. Finally, the Bit Level Interface interacts with the Bitstream class. This class manages the device bitstream and provides support for reading and writing bitstreams from and to files.

In addition, the Bitstream class can take data read back from hardware and map it to the underlying bitstream data format. This ability to manage readback data is necessary for dynamic reconfiguration. The *JBits* API uses the *XHWIF* software to download and readback from the hardware. *XHWIF*, in Xilinx hardware interface discussed later in this paper.

While this is all that is necessary to use the *JBits* interface, two other related pieces are included in Figure 2. The first in the Core Library. This is a collection of Java classes which define macrocells or Cores. These are usually parameterizable and relocatable within a device. Examples of Core are counters, adders, multipliers, constant multipliers and other standard logic and computation functions.

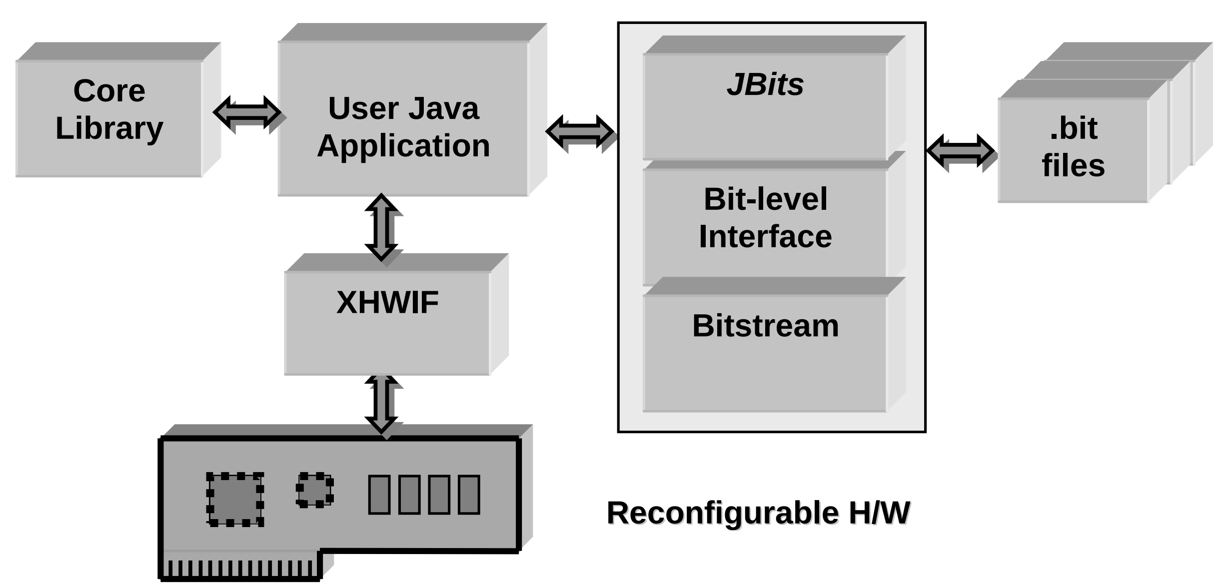

Fig 2. JBits System Design

A sample JBits code for the Virtex device is given in Fig 3.

```
/* Configure the F LUT of the Slice 0 of row, col to be XOR */
set(row, col, Slice0_FLUT, XOR);

/* Get the value of the Clock Input at row,col */
c = get(row, col, ClockInput);
```

Fig 3. Sample Code

4. LIMITATIONS OF *JBITS*

Perhaps the largest drawback of *JBits* API is its manual nature. Everything must be explicitly stated in the source code, including the routing. While this can become tedious, use of pre-constructed macrocells or Cores can greatly reduce this burden. It should be pointed out that this is also simply a function of the software versus the hardware model. Software requires complete specification of details, unlike automatic CAD tools. Related to this need for explicit specification of all resources, *JBits* interface favors more structured circuits. Unstructured circuits such as random logic are not well suited for direct implementation in *JBits* applications.

An equally important limitation is that *JBits* API requires that the user be very familiar with the architecture. While the Xilinx device architectures are actually completely documented in the Xilinx databook, most users have never had the need to learn such details. It is expected that the necessary understanding of the underlying device architecture will be the greatest barrier to the widespread acceptance of *JBits* interface, or any tool resembling *JBits* interface.

In addition, because *JBits* interface necessarily works at the bitstream configuration level, it exists at the most downstream end of the tool chain. While *JBits* API may make use of circuits produced by standard development tools, modification or reconfiguration of the circuit at the *JBits* level eliminates any possibility of using any analysis tools available to circuit designers further up the tool chain. Specifically, the ability to do any sort of timing analysis is absent in *JBits* software. It is not clear, however, that tasks such as timing analysis are even feasible in a dynamic reconfiguration environment. Small changes in the circuit configuration may have a dramatic impact on functionality as well as timing. It is not clear that the results of such changes to the circuit configuration can be predicted and analyzed in the general case. One tool which appears to have at least partially offset the lack of analysis tools is the recent development of *BoardScope* discussed in the next section.

4. BOARDSCOPE, THE HARDWARE DEBUGGER

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 uses the *JBits* interface to access resources in the FPGA's bitstream. Then using *XHWIF*, a portable hardware interface discussed later in the paper, the bitstreams are downloaded to configure the FPGAs, or readback to analyze them (see Figure 4). Currently, *BoardScope* supports devices in the Xilinx XC4000EX, XC4000XL and Virtex families.

BoardScope graphically displays the states of all CLB flip-flops for all computational FPGAs on the board. Figure 5 shows the *BoardScope* StateView for a *Pamette* board with four Xilinx XC4028EX FPGAs. Each major square represents 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 low state, while green denotes a high state. 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.

For a more detailed view of the circuit, the state of a Configurable Logic Block can be seen in the detailed CLB view(see Figure 5). The CLB view shows the 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 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 and Virtex architecture. The FPGAs in Figure 5 are loaded with a linear cellular automata demonstration circuit. The unique triangle pattern produced by such automata is visible on the display.

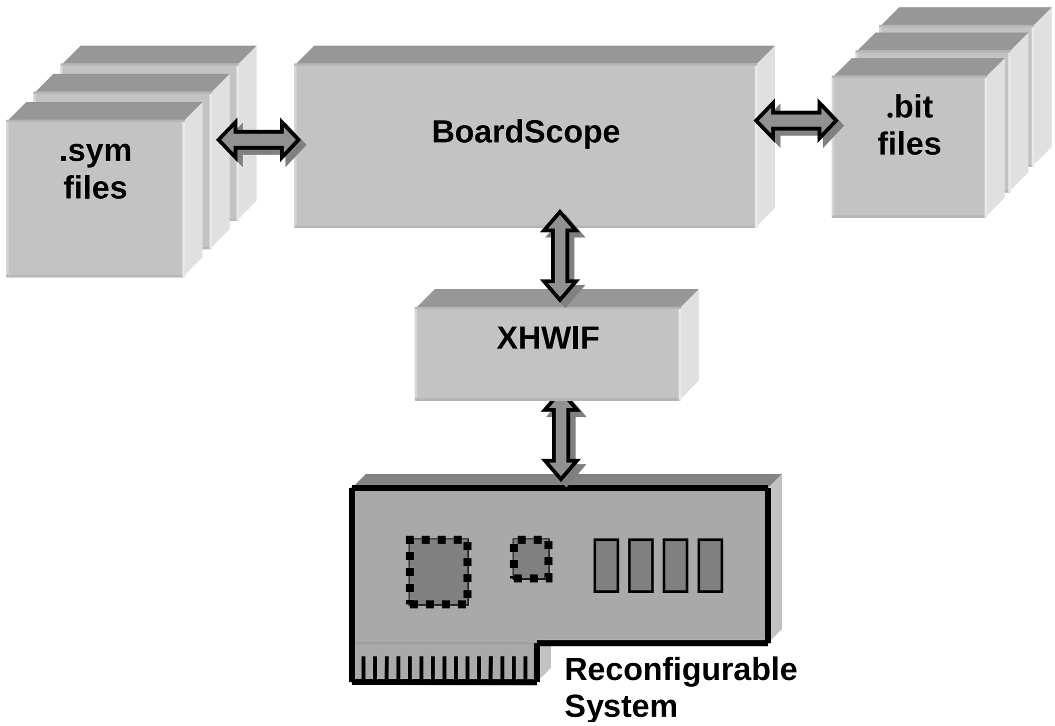

Fig 4. BoardScope Software Structure

5 THE USER INPUT INTERFACE AND SAMPLE FLOW

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. This button also clears the waveforms in the Signal View.

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 State View display and the Signal View displays. 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 **Zoom In** button magnifies the view of the State View and Core View displays. It does not affect the Signal View display.

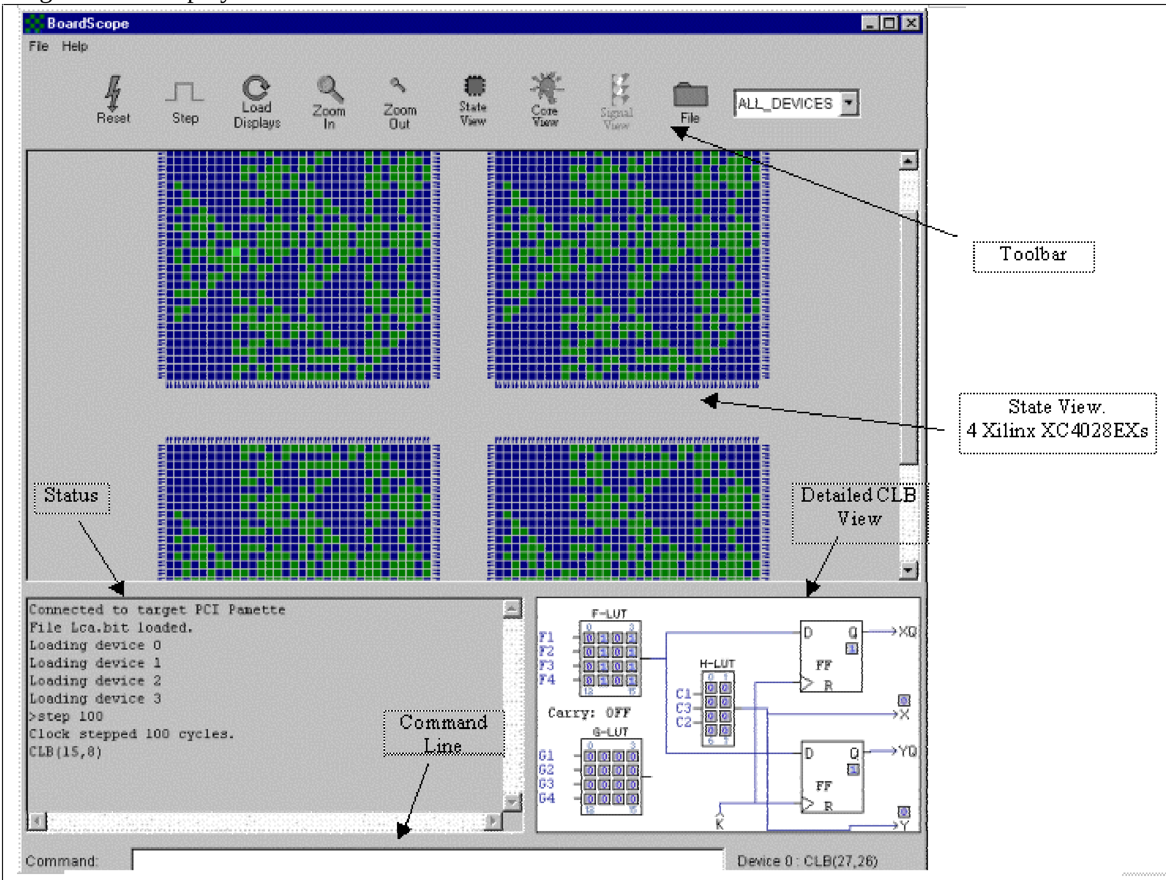

Fig 5 BoardScope Stateview Display

The **Zoom Out** button de-magnifies the view of the State View and Core View displays. It does not affect the Signal View display.

The **State View** button is used to show all of the CLB and IOB FPGA flip-flops in the Main Display.

The **Core View** button is used to show the layout of the JBits cores in the CLBs of the FPGAs in the Main Display.

The **Signal View** button is used to show the probe waveforms in the Main Display. This button is not enabled until a .sym file is loaded using the File button. A .sym file is a simple text file with the signal names present in the circuit under analysis.

The **File** button is used to load a .bit or .sym files. Loading a .bit file programs the FPGAs with the circuit described in the file. Loading a .sym file enables the Signal View display with the probes defined in the file. The devices targeted by the .bit or .sym file are based on the state of the Device Selector.

The **Device Selector** specifies which FPGA is to be configured if a .bit file is selected using the File dialog. If "All Devices" is showing, then all the FPGAs are configured with the selected .bit file. If a .sym file is selected using the File dialog, then the probe points are targeted to the device in the Device Selector (if the Device Selector specifies all devices, then the .sym file targets the 0th device).

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.

When BoardScope is first brought up, it shows a checkered State View 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 Core View button, the floorplan for the JBits cores are quickly examined. 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 State View button is clicked, revealing that several cores 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 other standards. 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 6). 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.

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 6 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: -XHWIFNet <machine name>

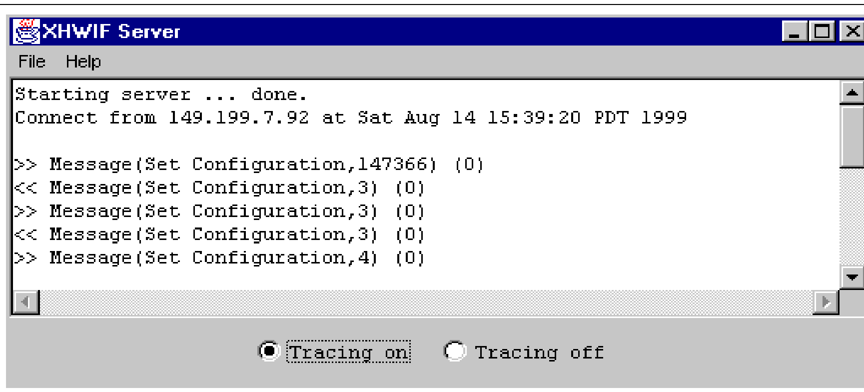

Fig 6 XHWIF Server

8 RUN-TIME RECONFIGURATION

Run-Time Reconfiguration (RTR) is defined as dynamically modifying the hardware circuits of the FPGA during the execution time. Run-time reconfiguration is achieved by integrating the *JBits* and *XHWIF* API in the *JBits* methodology. That is, the RTR application would make calls to *JBits* interface to modify the configuration data in the bitstream and would have to make explicit *XHWIF* calls to interact with the hardware. For example, the RTR application would make setConfiguration(device, data) and getConfiguration() calls to perform download and readback of the configuration data.

Figure 7 shows a typical *JBits* environment, where the *JBits* application would use *JBits* API calls to configure and modify the bitstream to specify a digital design. This design can be verified and debugged using BoardScope. Finally, the application can integrate the *JBits* and *XHWIF* calls to achieve RTR.

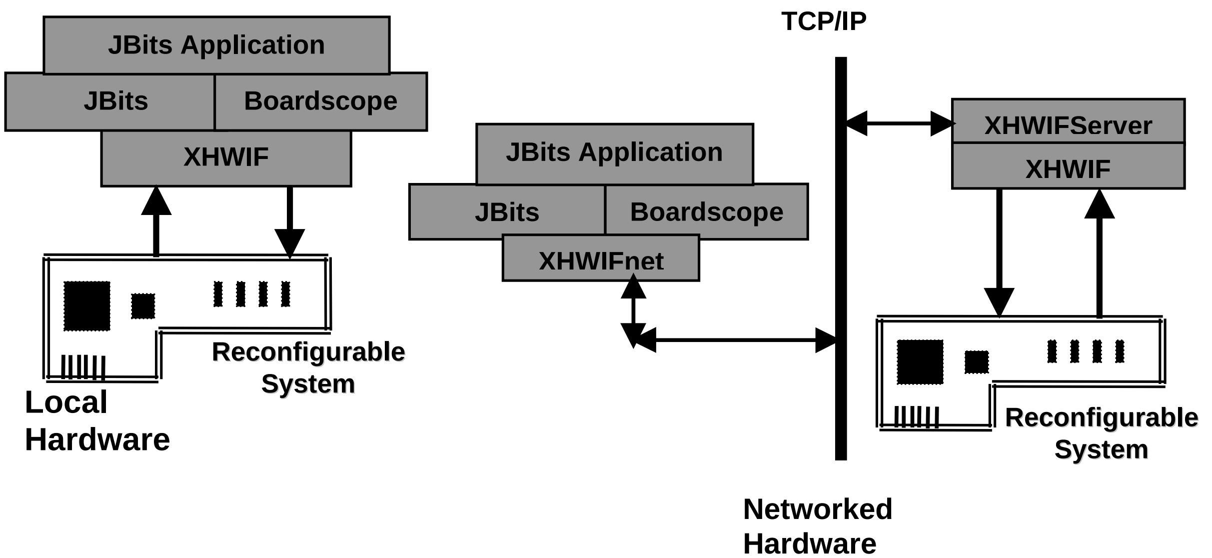

Fig 7 JBits Development Environment

9 CONCLUSIONS AND FURTURE WORK

The *JBits* API and its associated software, *BoardScope* and *XHWIF* provide a new methodology to design, develop, and verify digital circuits. The ability to produce circuitry in a true software development environment, with a quick edit / compile / debug cycle promises to change the way FPGA design is done. Large changes to designs can be made rapidly, bypassing the historically long run-times of traditional place and route CAD tools. The unification of the FPGA and the host code has enabled the development of dynamic run-time reconfigurable applications.

The capabilities and features of the *XHWIFServer* and *BoardScope* facilitates remote design development, debugging and even remote design deployment.

Future work for the *JBits* interface include the development of a routing API that would eliminate the manual routing. Also in plan is the development of a hardware simulator for design verification and an extensive parameterizable cores API.

10 REFERENCES

[1] Eric Lechner and Steven A. Guccione, "The Java Environment for Reconfigurable Computing", in Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications, FPL 1997. Lecture Notes in Computer Science 1304", Wayne Luk and Peter Y. K. Cheung, eds., Springer-Verlag, Berlin, September 1997, pp. 284-293.

[2] Xilinx, Inc., "XC6200 Development System Datasheet", 1997.
[3] Xilinx, Inc., "The Programmable Logic Data Book", 1996.
[4] Delon Levi and Steven A. Guccione, "*BoardScope*: A Debug Tool for Reconfigurable Systems, in Configurable Computing Technology and its use in High Performance Computing, DSP and Systems Engineering, Proc. SPIE Photonics East, John Schewel, ed., SPIE - The International Society for Optical Engineering, Bellingham WA, November 1998.
[5] S. A. Guccione and D. Levi, "Xilinx Bitstream Interface: A Java-based interface to FPGA hardware", in *Configurable 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.

JBits, XC4000, XC4000XL, VIRTEX, XHWIF, Xilinx Hardware Interface and XC6200 are trademarks of Xilinx, Inc. Java is a trademark of Sun Microsystems, Inc. All other trademarks are property of their respective owners.