Java Debug Hardware Models using JBits

Jonathan Ballagh[1], Peter Athanas[1], and Eric Keller[2]

[1] Virginia Tech
340 Whittemore Hall
Blacksburg, VA 24061
{jballagh, athanas}@vt.edu
[2] Xilinx Inc.
2300 55^{th} Street
Boulder, CO 80301
Eric.Keller@xilinx.com

Abstract. This paper presents a methodology for extending FPGA bitstream-level debug and simulation capabilities, through the inclusion of Java/JBits-based hardware device models. Using the JBits API, behavioral hardware models can be written in Java and used in simulations with the Virtex™ Device Simulator. Java lends the advantages typically associated with object-oriented design languages to FPGA bitstream-level debugging and simulation. This leads to more efficient design of hardware models as well as more flexibility in testing. Java also provides unique capabilities that assist interaction with the Java Debug Hardware Models (JDHM) during simulation. Also, the advantages of bitstream level design control and run-time reconfiguration capabilities provided by JBits offers benefits over traditional simulation techniques.

1 Introduction

There has been much research in the area of run-time reconfiguration (RTR) [11–13]. JBits is a tool that has created new opportunities in the RTR field by providing the ability to rapidly create and modify Xilinx® Virtex and XC4000™ FPGA circuitry at run time by allowing direct access into the configuration bitstream. When the JBits environment was first introduced, it required that debugging take place using physical hardware. Recently a device simulator was introduced that works on bitstreams to emulate the FPGA device in software. While this proved to be an important step, it didn't provide a way to simulate external hardware peripherals interacting with the FPGA. The drawback of the Virtex Device Simulator (VirtexDS), therefore, was that it did not provide an intrinsic scheme to bring data in and off of the device. This paper presents a methodology for designing Java based Debug Hardware Models (JDHM) that provide the needed I/O and external test bench control for the VirtexDS. The language constructs afforded by Java allow for object oriented design as well as run-time interaction with the modeled hardware object. These device models can range from RAM to a microprocessor. Using JDHM along with the Virtex Device Simulator, both static and run-time reconfigurable FPGA designs can be efficiently tested at the bitstream level. This approach provides a bitstream-level debug and simulation environment where no hardware is needed.

2 JBits

JBits [1] is a Java API that provides access to every configurable resource in a FPGA. Xilinx FPGAs [3] are SRAM based and have the ability to configure the device many times. JBits initially intended to provide a solution to support Run-Time Reconfiguration (RTR). RTR systems distinguish themselves by defining the circuits logic and routing just prior to, or during operation. RTR systems typically modify the circuit several times during the execution of the application. Because of the low-level design capabilities, JBits provides the necessary tools to efficiently modify or create a design. While JBits supports RTR, static designs are supported as well.

A typical JBits program creates logic in a new bitstream, modifies the logic and interconnects in an existing bitstream, or performs an analysis of a bitstream. The JBits API is built upon four main methods. The first two allow a bitstream to be read and written from a file. Another method allows the setting of a resource, i.e. a programmable interconnect point (PIP), to a certain value, i.e. on. The last method allows reading the state of a resource from the bitstream. The rest of the JBits API is a set of constants that define the configurable resources and their settings.

Communication can be done with an FPGA based board through the *XHWIF* API. The **X**ilinx **H**ard**w**are **I**nter**f**ace is an API that provides methods to communicate with FPGA based boards. This includes a method for reading bitstreams from the FPGAs. There are methods to write bitstreams to the FPGAs, step a clock, and write to and read from the memories on the board. Essentially, XHWIF provides a universal interface for communicating with different boards.

2.1 JBits environment

While JBits is a powerful design tool, designing at this level can be very difficult for all but the most trivial designs. Because of the object-oriented programming techniques afforded by Java, tools can be built upon JBits to provide higher abstraction levels. Tools such as JRoute [7], RTPCores [8], and JRTR [5] allow users the design control that is acceptable to them. JRoute is a router that has methods with varying levels of abstraction. A user can fully define each resource in a net, define a template to follow, or allow an auto-router do the work. The JBits run-time parameterizable (RTP) core specification provides a means for abstracting away the low level configuration calls, thereby creating an environment similar to traditional structural hardware design languages (HDLs) while concurrently affording the ability to make bitstream level modifications. The JRTR API is an extension of the JBits API to take advantage of the partial reconfiguration support provided by Virtex devices. This interface provides a caching model that automatically tracks changes to configuration data and only the modified data is written to or read back from the device.

3 Virtex Device Simulator

Another tool built on JBits is the *Virtex Device Simulator* (VirtexDS) [4]. The VirtexDS works directly from the FPGA bitstream using JBits to analyze the resource configurations of the device. Device level simulation has several advantages over traditional design simulators such as contemporary VHDL simulators. The first benefit is that it supports RTR. Other simulation environments address the issue of RTR. Lysaught proposes using Dynamic Circuit Switching to provide simulation support for RTR [10]. JDHL, a Java based hardware description language that defines circuits using objects, uses constructors and destructors to perform circuit reconfiguration during run-time [6]. Only JBits along with the VirtexDS, however, allow circuits to be synthesized and inserted into simulation at the bitstream level during execution. Using the JBits environment, circuit behavior during reconfiguration is accurately simulated as a result of the direct FPGA hardware emulation approach used by the VirtexDS. Device level simulation also has the advantage that it is implementation tool independent. The bitstream is a common format between all tools, similar to EDIF but at a lower level. Another advantage over traditional simulators comes from the fact that the VirtexDS implements the XHWIF API, thereby allowing users to transparently switch between the simulator and an actual FPGA.

3.1 External Stimuli

An environment has been set up to allow users to create stimuli on any pin inside of an FPGA by adding an event to the event queue in the VirtexDS. The pin can be any resource inside of the FPGA and is not limited to IOBs. For example, an input to a CLB can be set to a certain value. There is also the ability to read state information from flip-flops internal to the FPGA.

When the VirtexDS is initialized it creates a server object called SimulatorServer. Communication with the SimulatorServer object is accomplished using a client object called SimulatorClient. The communication is accomplished using a stream based socket network connection. While a TCP/IP connection is used, the client typically connects to the server process running locally, rather than across the network. However, the ability to communicate across a network is available if desired. The SimulatorClient provides the user with a method, *setPinValue(pin,value)*, which sets the given resource to the value. There is also a method *readPinValue(pin)* which returns the current value for that resource. Finally there is a method *waitForStep()* which waits for the VirtexDS to process all events during the current clock cycle.

4 External Hardware Device Model Design

The VirtexDS, in conjunction with the XHWIF interface, provides accurate models of the Xilinx Virtex device series. The inability of JBits to provide external stimulus into the chip has burdened the usage of the language for large application designs. This drawback is readily apparent in embedded system applications, in which the FPGA is connected to external devices, such as RAM and FIFOs. Ideally, the FPGA bitstream, including its interfacing to external peripherals, should be able to be tested for correct operation before it is loaded onto physical hardware.

The idea of full-system simulation can be achieved by extending the functionality of the SimulatorClient provided by the JBits API to correctly model the behavior of the selected hardware devices using the Java programming language. The characteristics of an external hardware peripheral can be described and encapsulated within a Java object that extends the SimulatorClient class. These objects are then used in conjunction with the VirtexDS, thereby expanding the capabilities of the simulator by providing the functionality afforded by the hardware models. Figure 1. shows the interaction between the hardware device model and VirtexDS in relation to a full JBits simulation environment. In using this approach, the capacity and size of these emulated external devices becomes limited only by the memory and storage of the users host computer, rather than the limitations imposed by physical hardware.

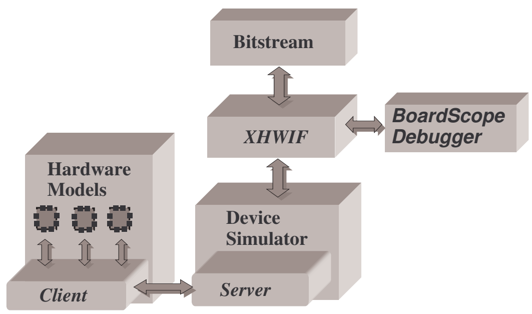

Fig. 1. Diagram showing a full system simulation environment using JBits.

An additional motivation for the creation of Java based peripheral models is illustrated in the situation in which a user would like to design or test a bitstream using an external peripheral that is currently unavailable or in the prototype stages. By first modeling the new device in software, the user has the ability to modify or design a new bitstream such that correct operation is guaranteed when the part finally becomes available. This can also be extended to situations where the FPGA to be used is unavailable.

4.1 Interfacing

The SimulatorClient class included in the JBits API provides a means in which the emulated hardware device models can communicate directly with the VirtexDS. This communication is accomplished using stream socket network connections. The SimulatorClient provides the functionality required to read and write logic states of individual Virtex FPGA pin resources using the *readPinValue(pin)* and *setPinValue(pin, int)* functions. By expanding on these functions, n-bit data vectors can be read or written to a defined pin array of length n. Vectors are read from the simulator using function *readVector(pin[])* and written using *writeVector(pin[], int)*. All hardware models require this basic functionality, and therefore should extend the SimulatorClient class.

At the top level of a JBits design the nets that connect to external hardware are passed to the JDHM object. The class that each JDHM object extends from has a method that translates each net to the pins that it connects to. Using that method to get what pins each net connects to, the JDHM object will then be able to communicate with the VirtexDS. When a circuit is reconfigured, the pins may change and the JDHM objects will need to be passed the new nets. For non-JBits designs, the JDHM objects are just passed a list of pins.

By using the functionality provided by the SimulatorClient, a hardware component can be modeled using Java, constrained in functionality only by the size of the I/O pin arrays. The inherent advantage lies in the fact that a specific core or a design with a reliance on external stimuli can be verified for correct functionality at the bitstream level using a Java application. Additional benefits of this approach include the flexibility for run-time user interaction with the hardware objects and support for run-time reconfiguration.

4.2 Operation

Bitstream-level verification can be accomplished by using the XHWIF API in conjunction with the VirtexDS and instances of all required external hardware models. A VirtexDS object must first be instantiated in order to create a model of the desired Virtex device. The device is then configured with the bitstream using the XHWIF API. At this time, an instance of each external hardware model must be created. Instantiating the device model calls the constructor of the super class, SimulatorClient. This in turn connects the device model to the simulator using a stream based socket connection. Completion of these steps establishes the full simulation environment.

Device readback and clock stepping is then done using XHWIF. Since chip I/O is done using the external hardware models, each model must be responsible for storing and analyzing its own I/O data according to that designs particular verification requirements. These steps are most easily accomplished using BoardScope [2] as the GUI in order to hide the XHWIF commands from the user. At this time, however, the external hardware models must be instantiated outside of BoardScope.

4.3 Bitstream/Core Design Debugging and Verification

Previously, FPGA debugging at the hardware level was encumbered by the “black box” nature of the configuration bitstream. The behavior of an individual signal could be traced at the hardware level by incorporating additional routing into the design before synthesis, in order to bring the signal to an unused IOB. This approach suffers in the sense that a free I/O resource may be unavailable, and that selecting a different signal to view requires an additional synthesis overhead. While device readback is often used for debugging, post processing of report files is often required to reconstruct logic placement.

Although BoardScope provides graphical insight into the functionality of the bitstream, the debugging process is encumbered by the lack of support for device I/O. While CLB logic can currently be programmed to generate input stimuli and control signals for debugging, this method is extremely wasteful in terms of resource consumption. This approach becomes extremely impractical when design operation involves streaming large amounts of data through the FPGA, as is often the case in signal processing applications. This approach also fails in cases where user interaction is desired.

An alternative approach, therefore, must be devised for bitstream design debugging and verification.

The concept of the external hardware model can be extended for debugging purposes. In essence, almost any test vector generation model can be applied to an existing bitstream, with the addition of a JDHM object. Through correct interfacing, control over all FPGA I/O can be passed to a Java application. During the simulation process, data read back from the VirtexDS can be compared against a set of desired output vectors for correctness. The Java application or hardware device model can inform the designer of any inconsistencies with corresponding error messages to the screen or output file.

Graphical user interfaces can be built onto the external device hardware models, such that when used together with BoardScope, provide for powerful system level graphical bitstream debugging environments. Since the hardware models are written in Java, a device customized GUI could easily be created using Java Swing or AWT classes to wrap the I/O controls afforded by the model. The complexity of the debugging process can be greatly simplified by using the hardware models GUI together with BoardScope. The result is software level debugging at the hardware level.

4.4 Possible Applications

External Memory External memory devices, such as RAM, ROM, and FIFO devices are easily described and modeled using Java and connected to the simulator. Memory contents can be loaded from a file at run-time or entered from a text prompt if desired.

The external memory model approach affords many advantages over physical hardware. Unlike their physical counterparts, selected memory addresses or blocks can be displayed to the screen or written to a file whenever desired. Also, support can be added so that the user can modify memory contents during operation if desired. This becomes extremely valuable in the debugging process. Also, data files of any length can be streamed through the bitstream design using an external FIFO model used to provide input into the device. An additional FIFO model would read output data from the device as it is streamed through and record it to the file or send it to another application for processing. As a result, lengthy simulations complete faster than traditional simulation tools as a result of the high speed nature of the VirtexDS.

Control Signals Simple control signals can be driven into the device through a simple device model. The device model would offer control over the signals through a GUI, text prompt, of file. For example, a GUI could be created with a *RESET* button that controlled the reset line of the device. Features could also be created that would allow for certain signals or buses to be enabled and disabled depending on the designers preference. This aids in isolating and identifying design errors.

Microprocessor Interfacing FPGAs are often used in conjunction with microprocessors and microcontrollers. If the internal processor can be ported to Java, the processor model can interact with the bitstream in the same manner as the physical hardware.

4.5 Comparison with Other Methods

The JDHM environment provides several benefits over traditional FPGA design simulation and debug applications. The principle benefits include design verification at the bitstream level, support for run-time reconfiguration, and flexibility in the simulation environment and controls. Using a standard HDL for test bench creation, such as VHDL, does allow for hardware modeling using a behavioral paradigm. However, that approach is at the design level and does not necessarily provide an accurate model of behavior once the design is run on physical hardware. The work flow that transforms VHDL or Verilog down to a bitstream is a long pathway filled with many steps. Furthermore, some of these steps are quite complex and are prone to errors, such as behavioral synthesis. As a result, the computational design specification is far removed from the process that generates the bitstream. Validation of a design in this traditional flow requires that each compilation step be error free, which is often impossible to accomplish. VHDL design and debug environments also do not support run-time reconfiguration and provide little in the way of flexibility. While other debug environments attempt to address debug of run-time reconfigurable systems, they fail in many regards. JHDL [6] provides an alternative to traditional HDLs, giving the designer device modeling and hardware level debugging capabilities [9] using Java. While JHDL originally intended to support RTR, it still relies on main-stream synthesis tools as a back end. Although JHDL provides effective hardware debugging capabilities, it requires an FPGA hardware platform that supports clock stepping and readback funtions. The VirtexDS, on the other hand, requires no hardware to operate, and can therefore be used to simulate and debug any Virtex configuration bitstream.

5 Conclusions and Future Work

An environment has been presented that demonstrates the use of Java as a language to model hardware and interact with the Virtex Device Simulator to debug complete FPGA-based systems. The JBits API was used to test the configuration bitstream design of a Virtex device. The external hardware was created with Java behavioral models that communicated with a device simulator through a stream based socket connection. Experimental embedded system designs using peripherals that are unavailable can be debugged and modified using the method of hardware models as described in this paper. Using Java has many advantages over a hardware description language such as VHDL. Among these advantages are flexibility and the ability to interact with the JBits API. The JBits API, unlike traditional hardware description languages, supports run-time reconfiguration as well as affording designers complete design control.

This environment is still under development and there are many improvements that can be made. Currently only one VirtexDS is able to be open at a time. This imposes a limit to only one chip per system. While memory usage could become an issue with several Virtex devices being tested simultaneously, it provides a more realistic model of the system. Using a network of computers, simulations could efficiently spawn out several VirtexDS objects. Also, the environment presented here only supports synchronous communication. The hardware models set stimulous and then wait for the clock on the FPGA to be stepped. Having asynchronous events could provide additional flexibility and a more accurate model for certain external devices. Work is being done to integrate JDHM with the VirtexDS more closely. Currently java sockets are used which could add an unnecessary overhead. Finally extensions to a GUI tool such as BoardScope would make for a more user friendly debug environment. This extension would support loading in a model of a system and instantiate the necessary hardware models.

Acknowledgements

This work was supported by DARPA in the Adaptive Computing Systems (ACS) program under contract DABT63-99-3-0004.

References

1. S. A. Guccione and D. Levi, "XBI: A Java-based interface to FPGA hardware," *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.
2. D. Levi and S. A. Guccione, "BoardScope: A Debug Tool for Reconfigurable Systems," *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.
3. Xilinx, Inc., *The Programmable Logic Data Book*, 1999.
4. S. McMillan, B. Blodget, and S. Guccione. "VirtexDS: A Virtex Device Simulator." To be presented at SPIE 2000, Boston MA, November 2000.
5. S. McMillan and S. Guccione, "Partial Run-Time Reconfiguration Using JRTR," *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications*, Lecture Notes in Computer Science 1896, 2000.
6. P. Bellows and B. Hutchings, "JHDL-An HDL for Reconfigurable Systems," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 1998.
7. E. Keller, "JRoute: A Run-Time Routing API for FPGA Hardware," *7th Reconfigurable Architectures Workshop*, Lecture Notes in Computer Science 1800, pp 874-881, Cancun, Mexico, May, 2000.
8. S. Guccione and D. Levi, "Run-Time Parameterizable Cores," *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*, Lecture Notes in Computer Science 1673, pp 215-222, 1999.
9. B. Hutchings, B. Nelson, and M. Wirthlin, "Designing and Debugging Custom Computing Applications," *IEEE Design & Test of Computers*, Volume 17 Issue: 1, pp 20-28, Jan.-Mar. 2000.
10. P. Lysaght and J. Stockwood, "A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays," *IEEE Transactions on Very Large Scale Integration of VLSI Systems*, pp 381-390, September 1996.
11. W. Luk, N. Shirazi and P. Cheung, "Compilation tools for run-time reconfigurable designs," *IEEE Symposium on FPGAs for Custom Computing Machines*, pp 56-65, April, 1997.
12. H. Schmit, "Incremental Reconfiguration for Pipelined Applications," *IEEE Symposium on FPGAs for Custom Computing Machines*, pp 47-55, April, 1997.
13. J. Burns, A. Donlin, J. Hogg, S. Singh, and M. deWit, "A Dynamic Reconfiguration Run-Time System," *IEEE Symposium on FPGAs for Custom Computing Machines*, pp 66-75, April, 1997.