JBits: A Java-Based Interface to FPGA Hardware

Steven A. Guccione and Delon Levi
2100 Logic Drive, San Jose, CA 95124
E-mail: Steven.Guccione@xilinx.com
E-mail: Delon.Levi@xilinx.com

ABSTRACT

JBits(tm), the Xilinx Bitstream Interface is a set of Java (tm) classes which provide an Application Program Interface (API) into the Xilinx FPGA bitstream. This interface operates on either bitstreams generated by Xilinx design tools, or on bitstreams read back from actual hardware. This provides the capability of designing, modifying and dynamically modifying circuits in Xilinx XC4000 (tm) series FPGA devices. The programming model used by *JBits* is a two dimensional array of Configurable Logic Blocks (CLBs). Each CLB is referenced by a row and column, and all configurable resources in the selected CLB may be set or probed. Additionally, control of all routing resources adjacent to the selected CLB are made available. Because the code is written in Java, compilation times are very fast, and because control is at the CLB level, bitstreams can typically be modified or generated in times on the order of one second or less. This 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.

Keywords: FPGA, reconfiguration, Java

1. INTRODUCTION

JBits, the Xilinx Bitstream Interface is a set of Java classes which provide an Application Program Interface (API) into the configuration bitstream for devices in the Xilinx XC4000EX (tm) and XC4000XL (tm) families. This interface permits all configurable resources in the device to be individually set 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*. This provides a simple alternative to existing CAD-based tools.

2. THE DESIGN FLOW

JBits is based on the Java Environment for Reconfigurable Computing for the XC6200 (tm) 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 (tm) family.

The original motivation for *JBits* was to support dynamic reconfiguration in the Xilinx XC4000 (tm) family of parts. While dynamic reconfiguration has always been possible in all 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 resemblence 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. A final barrier was the architecture of the Xilinx XC4000 series device. Unlike the XC6200 (tm) family which supported partial reconfiguration, the XC4000 (tm) family requires that the entire device be reconfigured. This implied that the tool would have to work with complete XC4000 (tm) configuration bitstreams and insert configuration data directly into the bitstream in an orderly manner to provide dynamic partial reconfiguration.

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.

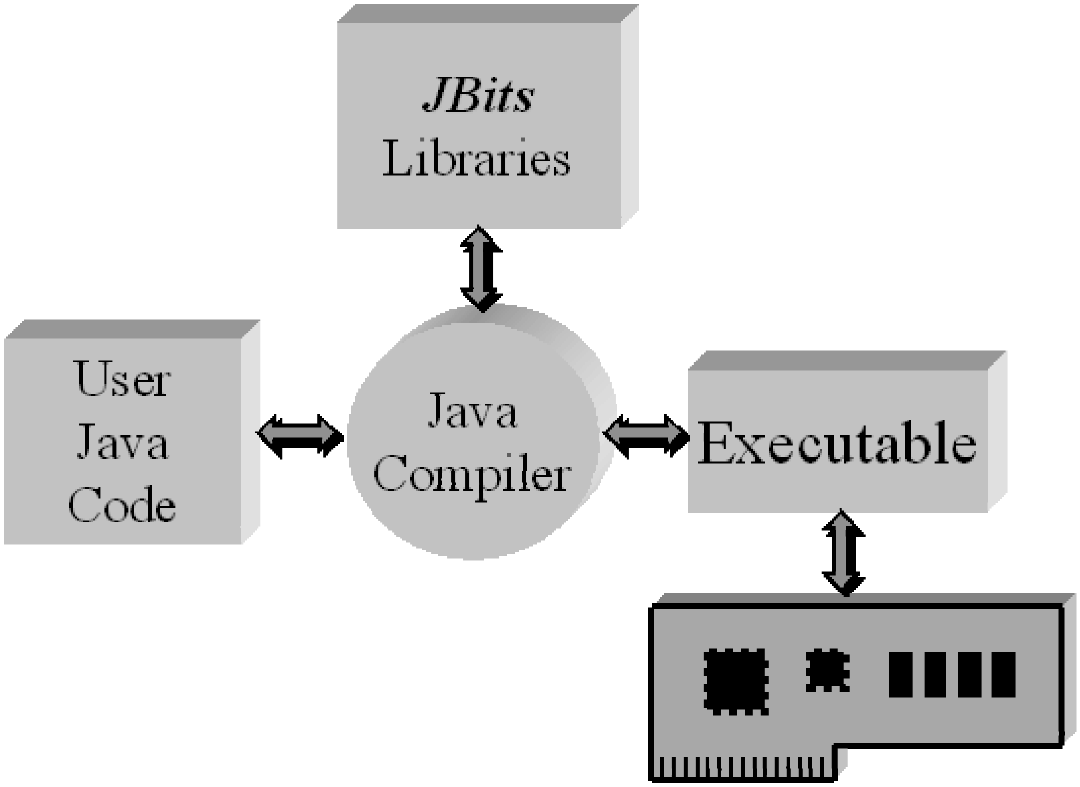

Figure 1: The *JBits* design flow.

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 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 XC4000 (tm) family configurable resources. 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 set 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 XC4000EX (tm) 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 live hardware and map it to the underlying bitstream data format. This ability to manage readback data is necessary for dynamic reconfiguration.

While this is all that is necessary to use *JBits*, 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.

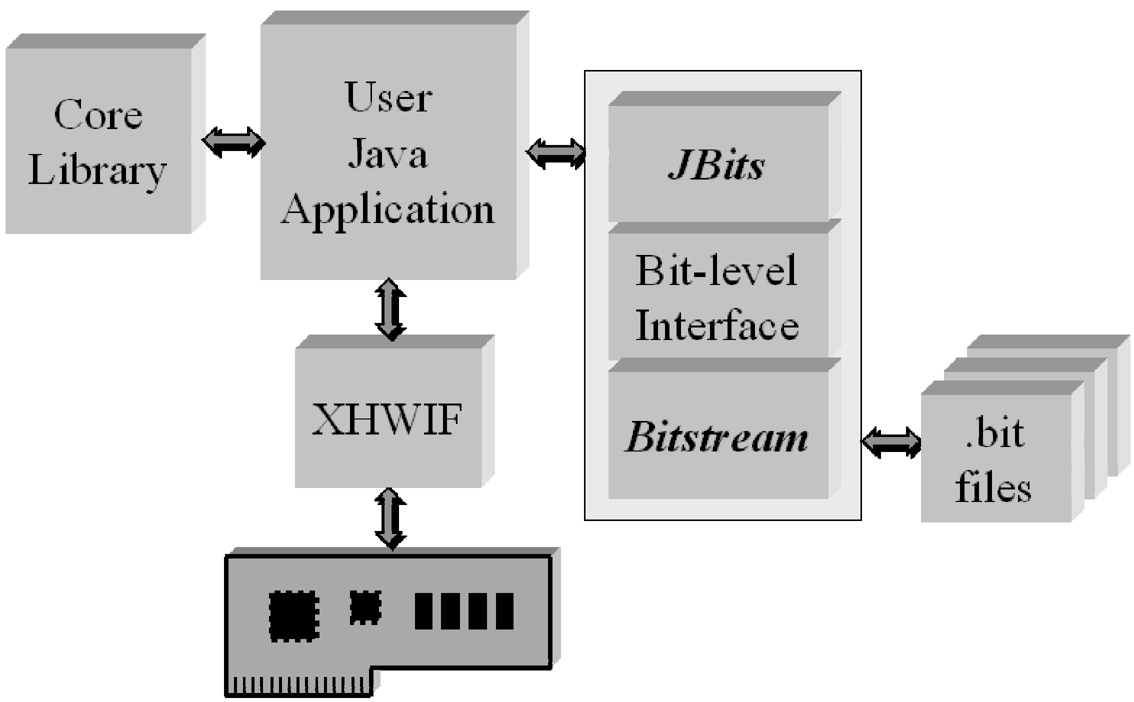

Figure 2: The *JBits* system.

Lastly, the Xilinx Hardware Interface (XHWIF) (tm) provides a simple, portable layer to connect *JBits* applications to FPGA-based hardare. Once this layer is supplied, *JBits* applications can run, without recompilation, on a variety of platforms. This interface has currently been ported to WildFire and WildForce hardware from Annapolis Microsystems, the Pamette hardware from the Digital Equipment Corporation and image processing hardware from GigaOps. Other ports are planned or are in progress.

4. A SIMPLE APPLICATION

The code in Figure 3 illustrates an application which produces a simple ripple carry counter. The approach taken is to produce a single CLB configuration which may be repeated in a loop to produce an arbitrary length counter. Note that all of the code is standard Java, with calls being made to the instantiated *JBits* object, *JBits*. The first obvious feature of *JBits* is the interface. Only a single *JBits* function call, set() is used to do all configuration. This function takes as its first two parameters a row and column, which define which CLB in the array is to be configured. Two other parameters define the configurable resource in the CLB and the value to which this resource is to be configured. In the case of the LUT configurations, for instance, the parameters are pre-defined constants which implement a desired boolean function.

Because there are so many configurable resources within the CLB, these resources are broken up in to several different Java objects. The Y output of the CLB, for instance is defined in the Y Java class. Within this class are constants giving all legal values to which the Y output, named Y.OUT may be set. In the code in Figure 3, the Y output Y.OUT is set to be the output of the G LUT, or Y.G.

The last 6 lines of code set the routing. At this point it is perhaps more instructive to look at a circuit diagram. Figure 3 below shows a single repeated cell which provides the basic counter functionality. The XOR gate in a feedback loop with the flip flop produces the equivalent of a toggle flip-flop and the AND gate supplies the toggle signal to the next stage. When all stages below are logic 1, the flip flop toggles. This provides the desired binary counting function. Note that connections between CLBs make use of shared routing resources, in this case the number 1 single length horizontal line and the number 1 single length vertical line. Note that many other routing resources are available in the XC4000 family devices, but in this particular application, only the two single length routing lines mentioned are used.

```
JBits  jBits = new JBits(Devices.XC4036EX);

col = START_COLUMN;
for (row=ROW_START; row<(ROW_START+size); row++) {

   /* Set F and G LUTs */
   if (row == ROW_START) {
      jBits.set(row, col, LUT.F, LUT.INVERTER1);
      jBits.set(row, col, LUT.G, LUT.BUFFER1);
      } else {
         jBits.set(row, col, LUT.F, XOR);
         jBits.set(row, col, LUT.G, AND);
         }

   /* Set Y output to output of G LUT */
   jBits.set(row, col, Y.OUT, Y.G);

   /* Set X flip flop inputs from F LUT */
   jBits.set(row, col, XFF.D, XFF.F);

   /* Set F2 and G2 inputs to Y output of */
   /* CLB below (via single wire) */
   jBits.set(row, col, Y.HORIZ_SINGLE1, Y.ON);
```

```
jBits.set(row, col, F2.F2, F2.HORIZ_SINGLE1);
jBits.set(row, col, G2.G2, G2.HORIZ_SINGLE1);

/* Set F1 and G1 inputs to XQ output of */
/* CLB (via single wire) */
jBits.set(row, col, XQ.VERT_SINGLE1, XQ.ON);
jBits.set(row, col, F1.F1, F1.VERT_SINGLE1);
jBits.set(row, col, G1.G1, G1.VERT_SINGLE1);

}  /* end for(col) */
```

Figure 3: A *JBits* ripple carry counter.

Another significant feature of this code is the structure. Note that the code consists of a single for() loop which sets CLBs in a single column. In addition, the row and column location of the counter are described by variables and may take any values. This permits this piece of code to be used to produce a variable sized counter at any location in the device. Simply converting this code fragment into a Java class object will produce a relocatable, run-time parameterizable counter core.

While not an integral part of *JBits*, this approach of using repeated cells to produce cores appears to be a generally useful technique for building structured circuits. Also note that in the code for the counter in the example, an if() statement is used to optimize the first stage of the counter. Here the XOR gate of the flip flop is reduced to an inverter and the carry propogation is reduced to a buffer. The alternative would have been to require an external toggle input control signal to the first stage of the counter. Note that this type of optimization may be used in other situations to produce circuits which are smaller, faster, or otherwise more efficient that an rigid repitition of a single CLB configuration. In general, this repeated cell approach is based on Kung's systolic VLSI design style.

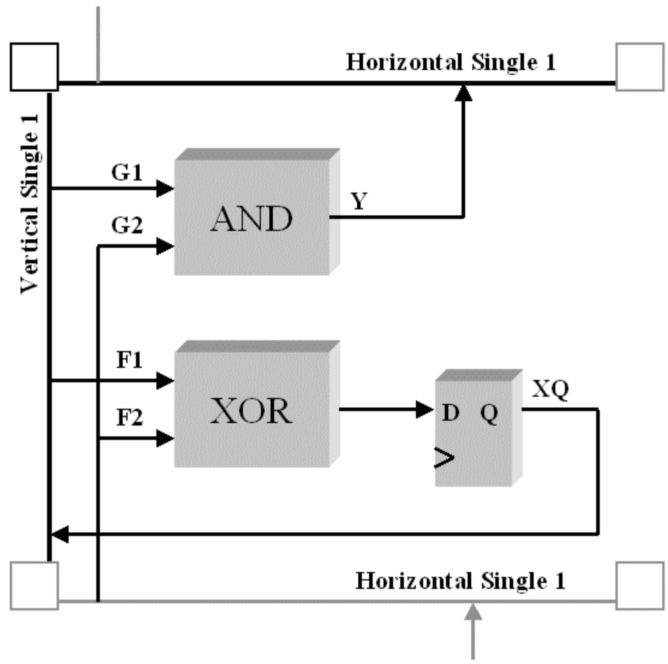

Figure 4: The counter cell circuit.

Finally, identifying and remembering the names of all of the resources and their legal values in a given architecture may seem challenging. But a sorted and index list of all resources and their legal values are documented in the *JBits* on-line documentation. This documentation is a set of HTML files which may be viewed by any standard Web browser.

5. LIMITATIONS OF *JBITS*

Perhaps the largest drawback of *JBits* 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* favors more structured circuits. Unstructured circuits such as random logic are not well suited for direct implementation in *JBits*.

An equally important limitation is that *JBits* 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*, or any tool resembling *JBits*.

In addition, because *JBits* necessarily works at the bitstream configuration level, it exists at the most downstream end of the tool chain. While *JBits* 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*. 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*. *BoardScope* is a *JBits* application which supplies an interactive point and click interface to FPGA-based hardware. With *BoardScope*, configured circuit state can be queried at any time and displayed in a variety of forms. In early development and use of *JBits* the availability of this debug environment was invaluable.

In spite of these limitations, early experiences with *JBits* indicate that it is a powerful and useful tool. The ability to design circuits in a true software development environment, as well as the ability to do dynamic reconfiguration currently make *JBits* unique. In addition, much of the difficulty in using *JBits* springs more from the rich set of features in the underlying architecture rather than from the software itself. Simpler and more programmer-friendly architectures would make a tool like *JBits* significantly easier to use.

6. CONCLUSIONS

JBits provides a powerful new software abstraction to the Xilinx XC4000 FPGA family. With this new abstraction, circuits can be rapidly produced, configured and reconfigured. This capability will for the first time allow researchers and commercial customers to experiment with dynamic reconfiguration using a mainstream FPGA device. In addition, *JBits* supplies a complete API to the device, which permits other tools, including traditional place and route, to be implemented directly for Xilinx hardware. This will give CAD researchers the ability to test and evaluate new algorithms and approaches for these devices.

Finally 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. Finally, the ability to produce encapsulated, parameterizable, object-oriented components promises to provide a new level of support for the production and sharing of intelluctual property.

An earlier version of this paper was published as: Steven A. Guccione and Delon Levi, "XBI: A Java-based Interface to FPGA Hardware”, 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, pp. 97-

102, November 1998.

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] H. T. Kung, "Why Systolic Architectures?", IEEE Computer, 15, pp. 37-46, January 1982.

[4] Xilinx, Inc., "The Programmable Logic Data Book", 1996.

[5] 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.

[6] Steven A. Guccione, "Programmamming Fine-Grained Reconfigurable Architectures", PhD Thesis, University of Texas at Austin, May 1995.