WebScope: A Circuit Debug Tool

Steven A. Guccione

Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124 (USA)
Steven.Guccione@xilinx.com

Abstract. *WebScope* is an interactive graphical tool used to probe and stimulate circuits in reconfigurable logic devices. *WebScope* is implemented using the *Java* programming language and features a graphical point and click user interface, remote access to hardware and a limited symbolic debug capability.

1 Introduction

Systems based on reconfigurable logic have been increasing in popularity. While various hardware platforms have been built [4] 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)* debugger for the *Enable++* system [6], the *KRONO* and *ShowRB* debugger for the *PAM* system [2], the *T2* debugger for the *Splash 2* system [1], the *Hardware Promela Debugger (HPDB)* for the *PROMELA* system [7], the *DISC debugger (DDB)* for the *DISC* system [3] and the *CALLAS* debugger for *CHAMELON* [5].

While this is a substantial number of systems, none are discussed in detail, nor are supporting documents on such debug environments referenced.

WebScope attempts to fill this gap in the software development environment for reconfigurable logic based systems. *WebScope* provides a graphical, interactive interface to aid in the debugging of designs. While it is tempting to refer to *WebScope* as a debugger, it actually interfaces to the system at the hardware level and provides no direct support for software-style debugging.

In this respect *WebScope* more closely resembles *In-Circuit Emulators* (*ICE*es) popular in microprocessor development environments. While this system operates primarily at the hardware level, this permits *WebScope* to operate independently of other design tools or software packages. This also enhances the portability of the tool.

2 The System Architecture

WebScope is implemented in the *Java* programming language. Its implementation permits *WebScope* to run on a variety of hosts, from PCs to workstations,

using the same small set of files. Additionally, *WebScope* may be run using local hardware, or remotely using hardware on another host. This is useful in situations where several users wish to share a board, or in cases where physical access to the hardware is not possible. To operate *WebScope* remotely, it is necessary to run a network server. This server negotiates network connections and provides the interface from the network to the physical hardware.

Finally, *WebScope* may be run as either a standalone application using a *Java* interpreter, or as an applet running from a Web browser such as *Netscape* or the *Microsoft Internet Explorer*.

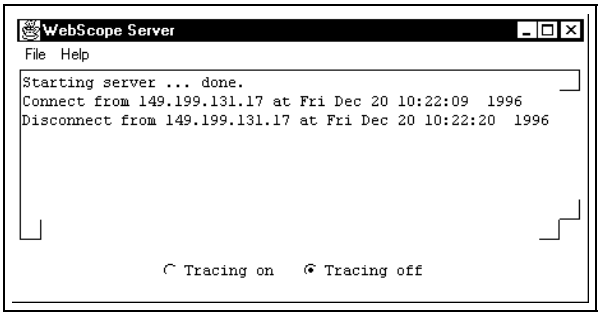

Fig. 1. The WebScope remote server.

The server supplied with *WebScope* is called *WsServer*. Like *WebScope* itself, this is a *Java* application. Figure 1 shows the server interface. All connections and disconnections to the hardware are logged, both to a window on the server and to a file. The *Tracing* option, when turned on, displays detailed information about the operations being performed remotely by *WebScope*.

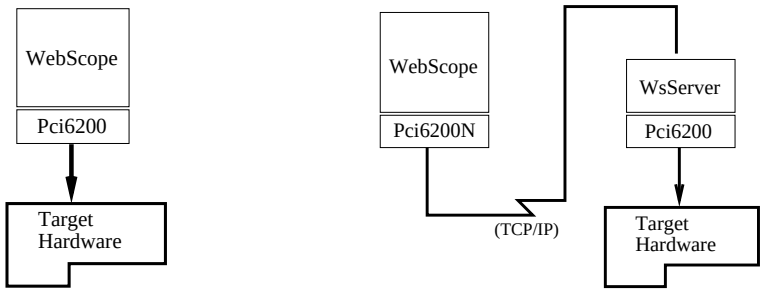

Fig. 2. Local and remote access to the target hardware.

Figure 2 shows the system architecture of *WebScope*. In this diagram, the

local and remote access modes are illustrated. All access to the hardware occurs through a well defined interface. Currently, *WebScope* operates with the *XC6200DS* [9] hardware but is designed to port easily to other systems using the *XC6200* device [8]. The *Pci6200* interface is used in both modes to communicate directly with the hardware.

In the networked version of this interface, *Pci6200N* provides the message passing interface to *WsServer*, the remote server. Note that in turn, the server uses the same *Pci6200* class as the direct connection mode to communicate with the hardware.

3 The Command Panel

The primary control provided by *WebScope* is via the row of buttons across the top of the display. When *WebScope* is initialized, only the **Connect** button is enabled. Only after successfully connecting to the target hardware will the other command buttons become active.

The next button, **disconnect**, is used to terminate the connection with the target hardware. Once *WebScope* is disconnected from the hardware, all functions, except for the **connect** button, are disabled. Note that the state of the hardware remains unchanged even after **Disconnect**ing from the target. Reconnecting will resume the session where it was left off.

The **reset** button is used to reset the hardware to its default state. The **step** button is used to advance the state of the system. This stepping sends a single clock pulse to the hardware. In addition, symbol table information is accessed. This is discussed in more detail in the section on the symbol table below.

The **reload** button is used to re-read all of the data from the target and re-display the result. This is particularly useful when some external software has modified the state of the hardware.

The **display** button is used to toggle *WebScope*'s primary display. The software architecture for the *WebScope* displays is flexible and permits custom displays to be easily added. Currently, there are three displays. These are:

- **Graphical display**: The array of cells (default)
- **Symbolic display**: The symbol table
- **Waveform display**: Strip chart style traces of the symbol table variables

These displays are discussed in more detail in the sections below.

Finally, the **file** button is used to load files into the *WebScope*. When this button is clicked, a dialog box requesting a file name is popped up. The type of file to be downloaded is selected by the buttons in the display. Currently, *XC6200 CAL* design files and *SYM* symbol files can be downloaded.

4 The Cell Display

When *WebScope* is initialized, the default display is the graphical display. For the *XC6200*, this display consists of a grid of 64 x 64 squares, representing the

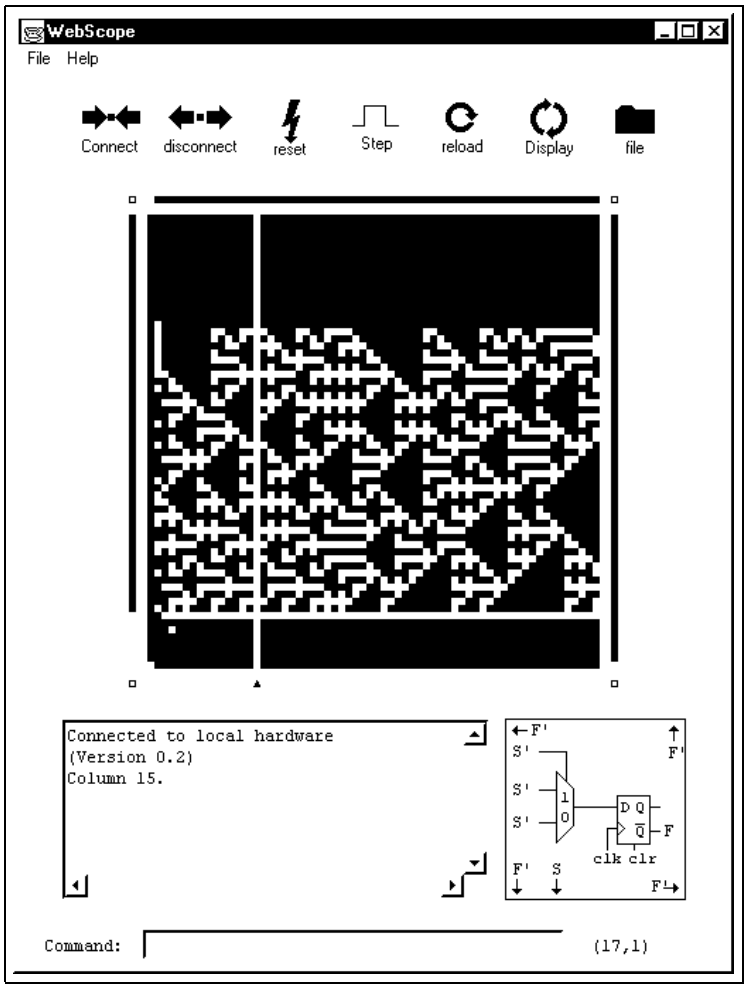

Fig. 3. The WebScope main display.

state of the logic cells in the *XC6200*. A green square represents a cell with a logic '1' output; a red square represents a logic '0'. The display in Figure 3 shows a linear cellular automata demonstration circuit. The unique "triangle" pattern produced by such automata is visible on the display.

The cell array is surrounded on its four sides by *bars*. The bar to the left of the cells represents the *Map register*. This register determines which cells in the *XC6200* are used during state accesses. Black cells indicate bits masked off by the Map Register, while white cells indicate enabled bits. Grey cells indicate bits that are enabled by the Map Register, but will not appear on the external bus, due to the current bus width setting. The bus width in Figure 3 is set to eight bits.

On the top of the cell array is the *Column Wildcard* register. This register determines which columns of the array are accessed during configuration. This permits multiple columns in the array to be written to simultaneously. Similarly, the *Row Wildcard* register is displayed as a bar on the right side of the array. Clicking on either wildcard register bar causes the value of the register to be printed in the *Status window* in the bottom portion of the display.

Below the cell array is a bar used to point to a given column in the array. Clicking on this bar highlights a column of cells in the array. Cells selected by the Map register are highlighted to help indicate their involvement in data accesses.

Fig. 4. The cell configuration status box.

Finally, clicking on cells in the array return their configuration value. This is displayed in the status panel at the lower right portion of the display. Note that this cell configuration panel has two modes. One displays textual information, the other displays a graphical representation of the circuit in the selected cell.

Clicking on the status panel toggles from one type of display to the other. Figure 4 shows the two modes for the cell configuration display. The graphical representation gives the four outputs of the cell (north, south, east and west), plus the extra "magic" output to the south. The three inputs to the primary cell multiplexer are also shown. In the textual display, the complete cell configuration is printed. This display is most useful to those intimately familiar with the *XC6200* cell.

5 The Symbolic Display

The second display in *WebScope* is the symbolic display. This display treats groups of cells in a column as a multi-bit variable and permits software-like symbolic access to the *XC6200* cells. Figure 5 shows the symbolic display used by the linear cellular automata demonstration circuit.

The variables are defined by the *name*, which is a symbolic name for the variable, a *column*, which indicates which column of cells in the *XC6200* is accessed, and a 64-bit bit pattern, specified by the *Map (high)* and *Map (low)* fields. These give the rows of the cells which combine to make the variable. Finally, the *Value* field displays the current value of the variable.

Name	Column	Map (high)	Map (low)	Value
clock	3	0xffffffff	0xffffff7f	0
clock	3	0xffffffff	0xffffff7f	1
random	5	0xffffffff	0x000000ff	0

Fig. 5. The symbolic display.

The symbol file format for *WebScope* is simply an ASCII text file containing the fields in the display, with an added field to indicate if the variable is a *read* or a *write* variable. *Read* variables are indicated by an "R", *write* variables by a "W". *Write* variables take a final parameter, the value to be written. This is only an initial value. It may be modified interactively from within *WebScope*. Figure 6 gives an example of a symbol file. Note that text following a hash character (#) are treated as comments and ignored.

```
#
#  Symbol file for WebScope
#  (for LCA demo)
#
# Name   Col      Map_high      Map_low    R/W     Value
#-----   ---      --------     --------    ---     -----
clock      3    0xffffffff   0xffffff7f      W        0
clock      3    0xffffffff   0xffffff7f      W        1
random     5    0xffffffff   0x000000ff      R
```

Fig. 6. The SYM file format.

On each **step** command, these variables are read or written in sequence, thus probing and stimulating the circuit. Note that in the example above, the same variable, *clock* is referenced twice, each time as a writable variable. This technique is used to produce a software-driven clock pulse on each **step**. Because the symbol table entries are read/written in the order in which they are listed, the cell at location (3,7) will be set to '0', then to '1'. This is used to clock the circuit in a software controlled manner.

6 The Waveform Display

The waveform display draws strip chart diagrams for the variables in the symbol table. This permits the history of the variable over time to be viewed. This

is not only useful for spotting transient irregularities in data, but for Digital Signal Processing (DSP) applications as well. This display may also be favored by hardware engineers and others who are more comfortable with the digital waveform displays found in circuit simulators.

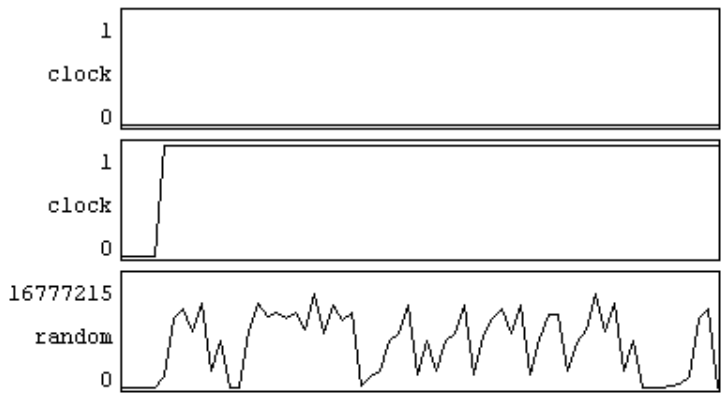

Fig. 7. The trace display.

As with the *Symbolic Display*, the *Waveform Display* is driven by the values in the symbol table. The example in Figure 7 shows the traces for the variables in symbol table in Figure 5. The value of the variable *random* contains bits in column 5, which is data produced by the circuit driving the linear cellular automata. These values are pseudorandom, as is indicated by the display.

7 The Command Line Interface

Finally, at the bottom of the screen is a command line interface. this interface permits complete access to the hardware. While not the preferred mode of operation, this interface is retained as legacy code from the original text-based interface to the *XC6200DS* hardware. This interface is still favored by some members of the design team.

On-line help is also available from the command line. This gives more detailed descriptions of the available commands and their syntax.

8 Conclusions

Perhaps just as significant as the tool itself is the process by which the software was developed. The original design contained a core functionality which simply displayed the state of cells in the *XC6200*. As the use of the tool increased, new features were requested and rapidly added. It is a testament to the power and

flexibility of the *Java* programming language and support libraries that this tool was able to be developed in such a manner in less than 6 man months.

In addition, the number of bugs was surprisingly low. This is not so much attributed to the skill of the programmer as to the extensive compile and run-time checking in *Java*. Most programming errors were identified quickly and repaired. Finally, the object oriented nature of the language enabled new functionality to be smoothly integrated into the existing body of code. Seldom was it necessary to modify existing code objects to add new functionality.

WebScope represents a new level of support for circuit and system development using reconfigurable logic. A powerful interactive interface to reconfigurable logic designs has already proven to be helpful in finding design errors at all levels of the system. It has been used to alternatively debug user designs, the development tools, system level hardware and even the silicon itself.

References

1. Jeffrey M. Arnold. The splash 2 software environment. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 88–101, Los Alamitos, CA, April 1993. IEEE Computer Society Press.
2. Patrice Bertin and Hervé Toutai. PAM programming environments: Practice and experience. In Duncan A. Buell and Kenneth L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 133–138, Los Alamitos, CA, April 1994. IEEE Computer Society Press.
3. David A. Clark and Brad L. Hutchings. Supporting FPGA microprocessors through retargetable software tools. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 195–203, Los Alamitos, CA, April 1996. IEEE Computer Society Press.
4. Steven A. Guccione. List of FPGA-based computing machines. World Wide Web page http://www.io.com/~guccione/HW_list.html, 1997.
5. Beat Heeb and Cuno Pfister. Chamelon: A workstation of a different colour. In Herbert Grünbacher and Reiner W. Hartenstein, editors, *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, pages 152–161, 1992. Proceedings of the 2nd International Workshop on Field-Programmable Logic and Applications, FPL 95. Lecture Notes in Computer Science 705.
6. H. Högl, A. Kugel, J. Ludvig, R. Manner, K. H. Noffz, and R. Zoz. Enable++: A second generation FPGA processor. In Peter Athanas and Kenneth L. Pocek, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 45–53, Los Alamitos, CA, April 1995. IEEE Computer Society Press.
7. Alan Wenban and Geoffrey Brown. A software development system for FPGA-based data acquisition systems. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 28–37, Los Alamitos, CA, April 1996. IEEE Computer Society Press.
8. Xilinx, Inc. *The Programmable Logic Data Book*, 1996.
9. Xilinx, Inc. *XC6200 Development System*, 1997.

This article was processed using the LaTeX macro package with LLNCS style