Debug of Reconfigurable Systems

Tim Price, Delon Levi and Steven A. Guccione

Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124 (USA)

ABSTRACT

While FPGA design tools have progressed steadily, availability of tools to aid in debug of FPGA-based systems has lagged. In particular, support for debug of run-time reconfigurable (RTR) systems have been all but absent. In this paper we describe *DDTScript*, a scripting language to aid in design, debug and test of RTR systems. The *DDTScript* language is fully integrated with the *BoardScope*^{tm} graphical debug tool and permits parameterization, instantiation, and removal of *Run-time Parameterizable (RTP) Cores* and other configurable circuit components. *DDTScript* also provides control of system level resources and supplies access to device state and configuration data. *DDTScript* is currently used not only to test and debug RTP Cores, but to construct and interact with complete designs. *DDTScript* is currently part of the *JBits*^{tm} tool suite and supports the Xilinx *Virtex*^{tm} family of FPGA devices.

Keywords: Debug, Regression Testing, Run-Time Reconfiguration, Virtex, Script, FPGA

1. INTRODUCTION

While FPGA design tools have progressed steadily, availability of tools to aid in debug of FPGA-based systems has lagged. Some early systems supplied simple debug aids as part of their development environments, but the debug tool portions of these systems were often thinly documented.[1–5] Recently, more sophisticated debug tools have found their way into mainstream software from FPGA vendors. While debug support for mainstream development appears to be receiving more attention, recent work in *Run-Time Reconfigurable (RTR)* circuits has provided new challenges for debug tools. Early work aimed at debug of RTR circuits[6,7] is actively being enhanced to provide support for these new debug activities.

One of the outstanding problems in debugging and testing such circuitry is that the number of possible configurations can be quite large. A *Run-Time Parameterizable (RTP)* core implementing a constant multiplier, for instance, can have parameters for bit width and for the multiplication constant. This provides a large number of possible instantiations, which are not practical to test exhaustively. The problem of testing circuits in this environment becomes more similar to that of testing software, where analysis and the testing of boundary conditions is the only practical approach. To better support this style of circuit debug and test, a scripting capability has been integrated into the existing RTR debug tools.

This scripting capability, *DDTScript*, permits circuits to be parameterized, instantiated and downloaded into hardware and tested. This testing can be performed using the standard debug tools and does not require additions to the existing design or new passes through the design tools. In addition, the ability to save scripting commands in external files and run them in a batch mode provides a way to perform large numbers of tests and to perform regression tests on modified circuits. This capability is useful in producing reliable circuits and libraries.

2. OVERVIEW

DDTScript was built using the *JBits* tool suite from Xilinx.[8] The *JBits* tool suite is a set of Java classes which provide an application programmer interface (API) into the configuration bitstream for the Xilinx *Virtex*^{tm} family of FPGA devices. All configurable resources in these devices can be individually programmed using this interface.

Run-Time Parameterizable (RTP) Cores are parameterizable logic cores that are constructed and instantiated at run-time.[9] These circuits can be configured and reconfigured based on information supplied in real-time by user

Further author information: (Send correspondence to Tim Price)
Tim Price: E-mail: Tim.Price@xilinx.com
Delon Levi: E-mail: Delon.Levi@xilinx.com
Steven A. Guccione: E-mail: Steven.Guccione@xilinx.com

software, user input or sensor data. RTP Cores utilize a certain number of Configurable Logic Blocks (CLBs) on the device when created and contain a *state* variable which helps track the operation of the circuit. For example, a *counter* core might designate its output bits in the *state* variable. This would permit the state of the counter circuit to be easily accessed and examined during operation.

Rather than attempting to manipulate FPGA resources directly, *DDTScript* uses the RTP Core abstraction as the primary interface to the hardware. As a practical decision, *DDTScript* is targetted at debugging RTP Cores and sytems composed to RTP Cores, so this level of granularity is appropriate. Additionally, using a higher level abstratcion such as an RTP Core permits *DDTScript* to be architecture independent. Designs using different FPGA device architectures and different RTP Core implementations can use the same *DDTScript* commands and scripts for design, debug and test.

While *DDTScript* can be used as a stand-alone text-based tool, it is also integrated into the *BoardScope* interactive graphical debug tool. *BoardScope* supports the single and multiple stepping of the system clock and interactive probing of FPGA resources. It also displays RTP Core layout information as well as graphical and waveform state data. Like all of the tools in *JBits*, *DDTScript* interfaces to hardware and hardware simulators via the *XHWIF*^{tm} interface.[7] This interface provides portable access to Virtex hardware and simulation environments. The use of *XHWIF* allows applications to be moved to different platforms, typically without recompilation.

3. DDTSCRIPT

DDTScript is an interactive, RTP Core-based command-line tool with a relatively simple syntax. *DDTScript* can be used to parameterize, intstantiate, remove and clock RTP Cores on FPGA devices in real time. Using *DDTScript* does not require that designs be recompiled or in any way passed through design tools. It is integrated as a command-line interface in the *BoardScope* debugger and it can alse run as a stand-alone command-line application.

The commands used by *DDTScript* require no specific knowledge of hardware description or programming languages. Table 1 shows the complete list of commands supported by *DDTScript*. The syntax is always the command name followed by parameters. Not all commands require parameters. If an incompletely parameterized command is issued by the user, the correct command usage is displayed.

Table 1. *DDTScript* commands.

RESET	Resets board and bitstreams
LOAD	Loads a bitstream file
CONFIGURE	Configures FPGA device with bitstream
ADD	Adds Core to bitstream
REMOVE	Removes Core from bitstream
READBACK	Reads back device data into bitstream
STEP	Steps board clock
DEVICES	Lists devices and types
SELECT	Selects FPGA device on board
LIST	Lists available Cores, constructors and variables
SET	Sets user variables
STATUS	Displays status of added Cores
WRITE	Writes out a static bitstream file
HELP	Displays list of commands
EXIT	Exits the stand-alone application.

Some *DDTScript* commands interact directly with the hardware while others are used for control and to provide debug information. The first group of commands in Table 1 all effect the hardware. The second group of *DDTScript* commands are used to control and interact with the debug environment.

Constructing and testing a circuit using *DDTScript* follows a process similar to other design and debug environments. First, a circuit is constructed by instantiating one or more RTP Cores to an FPGA configuration bitstream using the *ADD* command. This configuration bitstream is downloaded to the FPGA device or simulation environment with the *CONFIGURE* command. The circuit may then be clocked with the *STEP* command and state data read back using the *READBACK* command. This state data may be displayed using the *STATUS* command. These results can then be processed and compared to expected values. Script files can be created and piped into *DDTScript* from the command-line. This permits automation of the testing process, allowing cores to be tested with many different inputs.

Finally, *DDTScript* also has the ability to be used as a simple design tool. Once circuits are constructed, *DDTScript* can write out static bitstream files with the *WRITE* command. These bitstream files are snapshots of the current FPGA configuration and can be used to configure another FPGA device at a later time. Because these files are standard Xilinx confoguration bitstreams they may be used to interface with other existing tools and systems just as ony other bitstream generated with the mainstream design tools.

4. DEBUGGING A MULTIPLIER

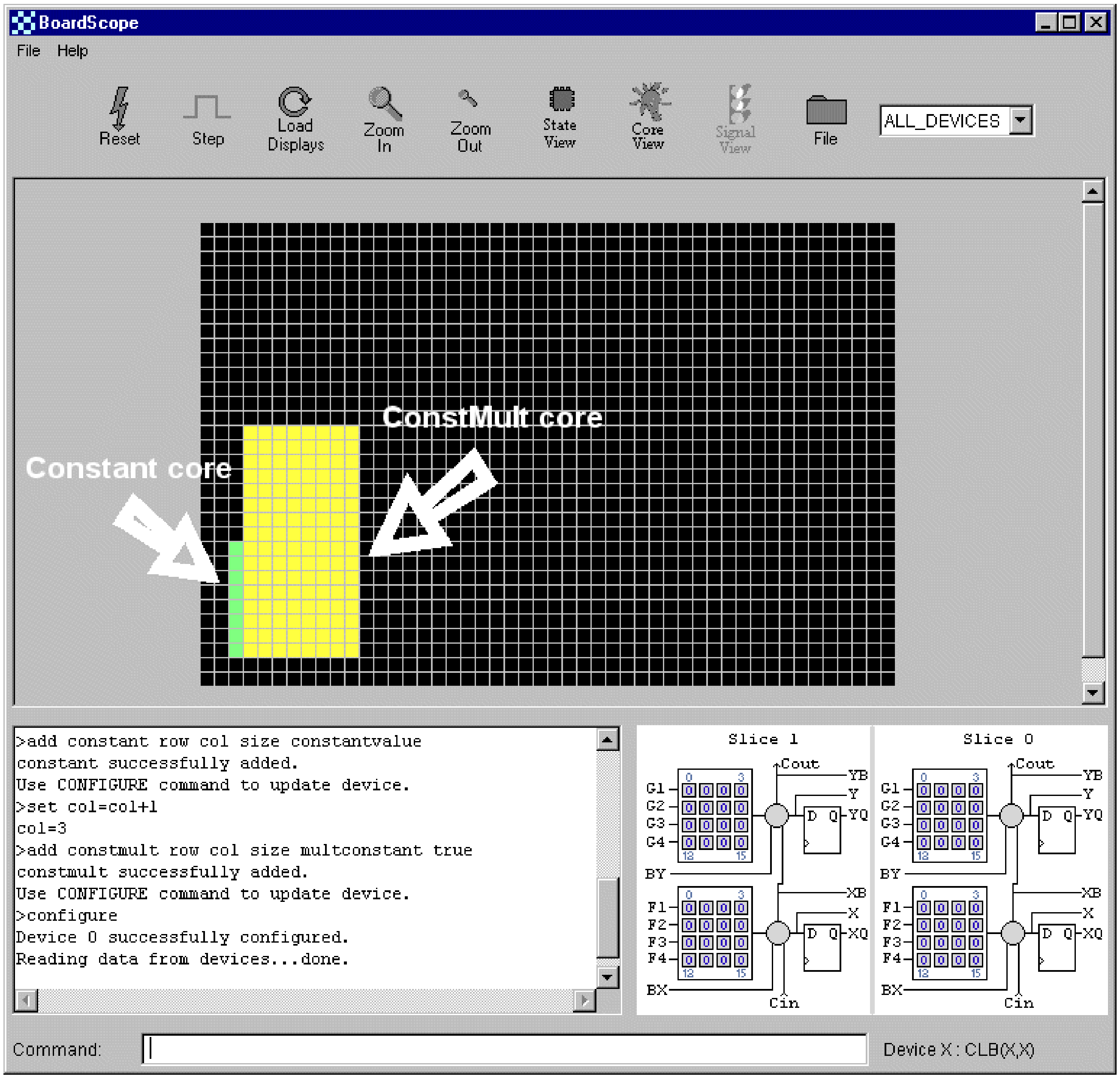
This section demonstrates how an RTP Core can be tested and debugged using *DDTScript* and the *BoardScope* debug tool. Figure 1 shows the script file on which this example is based. A constant multiplier core will be exercised with a test input and the result reported.

```
# Set up variables
set row=2
set col=2
set size=8
set steps=size+1
set constantvalue=91
set multconstant=243

# Reset the board and load in a bitstream
reset
load ../data/null300822.bit

# Add a constant valued core and
# place the constant multiplier beside it.
add constant row col size constantvalue
set col=col+1
add constmult row col size multconstant true
configure

# Step the clock
step steps

# Readback the data and display the results
readback
status
```

Figure 1. *DDTScript* script file to test a multiplier.

This script begins with several *SET* commands. These use used to initialize local variables used by the script. The names of these variables are user-defined and may appear anywhere in the script. *Row* and *col* are parameters which represent the coordinates of the origin CLB for the circuit. *Size* is another circuit parameter which represents the number of CLBs that the circuit will occupy. Other variables include the number of steps the FPGA device clock should be cycled and the constant parameter values for the cores. While the use of named variables is recommended and greatly enhances the clarity of the script, *DDTScript* accepts integer values and expressions for parameters as well as user variables.

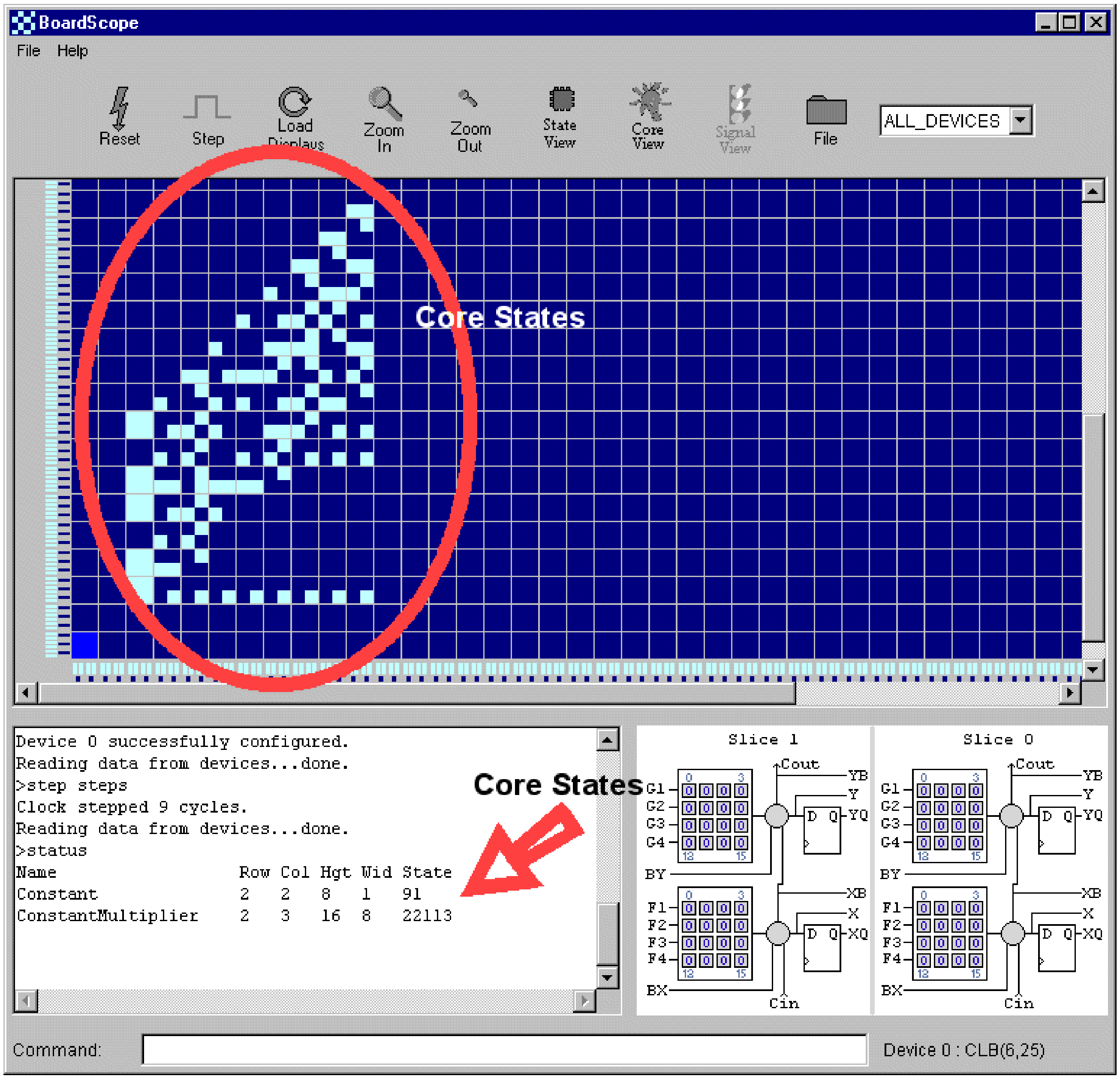
The hardware or simulation test environment is then reset using the *RESET* command and and an initial configuration bitstream is loaded into the system using the *LOAD* command. This particular bitstream file contains no logic or routing resources and is a "clean slate" used to build new circuits. It is also possible to use configuration bitstreams containing configured circuits that may be useful during testing.



```
>add constant row col size constantvalue
constant successfully added.
Use CONFIGURE command to update device.
>set col=col+1
col=3
>add constmult row col size multconstant true
constmult successfully added.
Use CONFIGURE command to update device.
>configure
Device 0 successfully configured.
Reading data from devices...done.
```

Figure 2. *BoardScope* debugger Core View showing the added cores.

Once this initial configuration bitstream is loaded, the cores may be added. The first RTP Core is a constant value Core which is used to supply inputs to the constant multiplier. The *constant* core is created with parameters that include the number of CLBs occupied, the constant value and the CLB coordinates of the Core origin. The second RTP Core created is the constant valued multiplier, *constmult*. This is placed one CLB column to the right of the *constant* core, as indicated by the incremented *col* variable.



```
Device 0 successfully configured.
Reading data from devices...done.
>step steps
Clock stepped 9 cycles.
Reading data from devices...done.
>status
Name                  Row Col Hgt Wid State
Constant              2   2   8   1   91
ConstantMultiplier    2   3   16  8   22113
```

Figure 3. *BoardScope* debugger State View showing result of multiplication.

As indicated by the parameters to the RTP Cores, the configured circuit now consists of a multiplier which multiplies by 243 connected to a constant test input core with a value of 91. Once this circuit is successfully instantiated, the *CONFIGURE* command is issued to download the configuration bitstream to the device. This configuration data is downloaded to device 0 in the system under test. By default, *DDTScript* selects device 0 of the board. If the system contains multiple devices, the *SELECT* command may be used to select the device currently under test.

Figure 2 shows the the configured circuit containing the constant and the constant multiplier RTP cores in the *BoardScope* graphical core view. *DDTScript* commands can be seen in the status area.

To exercise the multiplier core, the device clock needs to be stepped. The *STEP* command cycles the device clock for the requested number of steps. The constant multiplier core is pipelined and requires a number of steps equal to the number of bits in the test input. The extra clock step is necessary to initialize the device capture flip-flops that are used to capture state data.

The result of the multiplication is found by reading back the device data and asking for the *constmult* core status. Figure 3 shows the result of the multiply in the *BoardScope* state view. The *STATUS* command displays the state of each of the cores. As expected, the multiplier has a state value of 22113, which is 91 times 243. Had the actual result not matched what was expected, the *BoardScope* debugger could then be used to probe and examine the circuit.

5. CONCLUSIONS AND FUTURE WORK

DDTScript provides simple, effective design, debug and test support for run-time reconfigurable systems. Because *DDTScript* uses a small number of simple commands, core-based designs can be quickly generated and tested even during reconfiguration. This is especially useful for testing of RTP Core libraries. Test scripts can be produced and used to produce a test suite useful for evaluationa and regression testing. And because *DDTScript* operates at the Core level, scripts are architecture independent. Given equivalent core libraries, *DDTScript* scripts should run unmodified on a variety of architectures.

Future work includes adding direct routing support. Rather than requiring RTP Cores to provide core interconnection, future versions of *DDTScript* will have direct access to the routing API. Plans also include support for partial reconfiguration to reduce communication overheads with hardware. Finally, support for new Xilinx FPGA families is expected.

Both the *BoardScope* debug tool and *DDTScript* are available with the Xilinx *JBits* tool suite.

ACKNOWLEDGMENTS

This work was partially funded by DARPA under contract DABT-63-99-3-004 in the Adaptive Computing Systems (ACS) program.

REFERENCES

1. H. Högl, A. Kugel, J. Ludvig, R. Manner, K. Noffz, and R. Zoz, "Enable++: A second generation FPGA processor," in *IEEE Symposium on FPGAs for Custom Computing Machines*, P. Athanas and K. L. Pocek, eds., pp. 45–53, IEEE Computer Society Press, (Los Alamitos, CA), April 1995.
2. P. Bertin and H. Toutai, "PAM programming environments: Practice and experience," in *IEEE Workshop on FPGAs for Custom Computing Machines*, D. A. Buell and K. L. Pocek, eds., pp. 133–138, IEEE Computer Society Press, (Los Alamitos, CA), April 1994.
3. J. M. Arnold, "The splash 2 software environment," in *IEEE Workshop on FPGAs for Custom Computing Machines*, D. A. Buell and K. L. Pocek, eds., pp. 88–101, IEEE Computer Society Press, (Los Alamitos, CA), April 1993.
4. D. A. Clark and B. L. Hutchings, "Supporting FPGA microprocessors through retargetable software tools," in *IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, eds., pp. 195–203, IEEE Computer Society Press, (Los Alamitos, CA), April 1996.
5. B. Heeb and C. Pfister, "Chamelon: A workstation of a different colour," in *Field-Programmable Gate Arrays: Architectures and Tools for Rapid Prototyping*, H. Grünbacher and R. W. Hartenstein, eds., pp. 152–161, 1992. Proceedings of the 2nd International Workshop on Field-Programmable Logic and Applications, FPL 95. Lecture Notes in Computer Science 705.
6. S. A. Guccione, "WebScope: A circuit debug tool," in *Field-Programmable Logic: From FPGAs to Computing Paradigm*, R. W. Hartenstein and A. Keevallik, eds., pp. 308–315, Springer-Verlag, Berlin, August/September 1998. Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications, FPL 1998. Lecture Notes in Computer Science 1482.

7. D. Levi and S. 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*, J. Schewel, ed., SPIE – The International Society for Optical Engineering, (Bellingham, WA), November 1998.
8. S. A. Guccione and D. Levi, “XBI: A java-based interface to FPGA hardware,” in *Configurable Computing: Technology and Applications, Proc. SPIE 3526*, J. Schewel, ed., pp. 97–102, SPIE – The International Society for Optical Engineering, (Bellingham, WA), November 1998.
9. S. A. Guccione and D. Levi, “Run-time parameterizable cores,” in *Field-Programmable Logic and Applications*, P. Lysaght, J. Irvine, and R. W. Hartenstein, eds., pp. 215–222, Springer-Verlag, Berlin, August/September 1999. Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications, FPL 1999. Lecture Notes in Computer Science 1673.
10. Xilinx, Inc., *The Programmable Logic Data Book*, 1999.