Hardware/Software Co-debugging for Reconfigurable Computing

Karen A. Tomko and Anurag Tiwari

Department Electrical and Computer
Engineering and Computer Science
University of Cincinnati
Cincinnati, OH
ktomko@ececs.uc.edu

Abstract

Application development environments for Reconfigurable Computing are the topic of many research and development projects yet few comprehensive debugging tools have been provided. In this paper we describe a debugging environment for use with FPGA accelerated applications which supports co-validation and co-testing of the software and hardware portions of the application. Our Co-debugging environment supports in-situ debugging utilizing the readback capabilities of FPGA chips for fast recreation and isolation of a fault. We show that this environment has the potential to reduce application debug times from hours to just a few minutes.

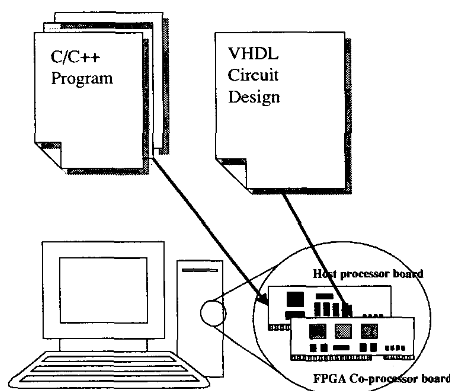

Figure 1. Reconfigurable Computing Application Model

1. Introduction

The typical Reconfigurable Computing application is initially developed in a pure software environment and is represented in a high level language such as C/C++. Implementation on a Reconfigurable Computing system is taken on to improve application performance after the original program has been validated. The Reconfigurable Computing application is partitioned into sections some of which run on a traditional general-purpose processor and some of which are implemented in FPGA hardware on a co-processor board as shown in Figure 1.

2. The Co-debugging Environment

We have developed an FPGA design verification technique which bridges the software and hardware debugging environments and shortens the time for each debug iteration by reducing the time it takes for simulation. Our Co-debugging environment supports in-situ debugging for fast recreation and isolation of a fault. Once the fault has been localized to a specific subcircuit, a given set of inputs, and a range of clock cycles, timing-accurate simulation is used to precisely locate the cause of the fault. Thus simulation time can be kept small by limiting the simulation to a subcircuit of the original design and a small window of clock cycles. This technique uses the readback and clock stepping capabilities of chips such as the Xilinx FPGA XC4000 series and Virtex series. The circuit state obtained via readback can be used to initialize the circuit signal values for a simulator such as the Modelsim HDL Simulator [8] to *fast forward* the simulation. Thus simulation can start just prior to the occurrence of the fault. See Figure 2 for an overview of the Co-debugging Environment.

The testing is carried out by running the host program under a software debugging environment. Prior to initiating execution of the FPGA circuit, a GUI is invoked which allows the user to step the clock, read the contents of memory

0-7695-0786-7/00 $10.00 © 2000 IEEE

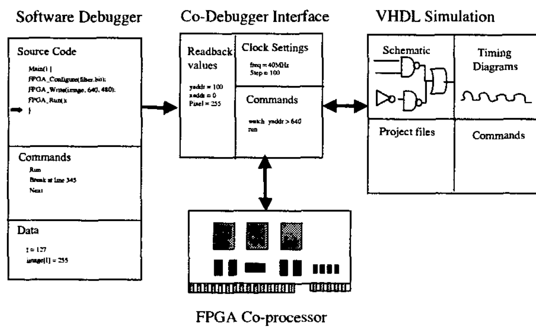

Figure 2. Co-Debugging Environment for Reconfigurable Computing

both on and off the FPGA chip, and to set up conditional trigger points. The board clock counter can be used to determine at what cycle the error occurred and the circuit can be reset and executed again for any specified number of clock cycles, allowing the user to stop execution at some point prior to the fault so that more diagnostic data can be gathered. Full readback of the chip can be used to determine the state of the circuit prior to the error. If the problem can not be determined based on the information available via readback, then the values which have been retrieved from the board are forced into the design simulation and a functional or timing-level simulation can be started so that a more detailed view of the circuit can be examined. The advantage in this kind of scheme is that the simulation does not have to be started from the very beginning, instead simulation starts just prior to the fault greatly reducing simulation time.

2.1. In-Situ Debugging Utility

The debugging utility has a graphical user interface implemented in Tcl/TK for interactive debugging of FPGA designs on an AMS Wildforce board. This interface can be invoked from within the host C or C++ program. It has the following capabilities:

1. The user can reconfigure any of the FPGA chips on the Wildforce board with the user specified bitfile.

2. The user can reset any of the FPGA devices on the board.

3. The user can specify the symbol file which maps the signals in the design to FPGA chip locations.

4. The entries from the specified symbol file are displayed in a scrollable window and the user can select a set of signals for which values will be displayed after readback. Both single bit nets and and multibit busses can be selected.

```
procedure Debug_ClkFreeRun ( )
{
1:      if debug_fpga = true then
2:         /* Call Tcl/TK routine to invoke the Debug Utility */
3:      else
4:         WF4_ClkFreeRun(···)
}
```

Figure 3. Software interface with In-Situ Debugging Utility

5. The user can control the clock by setting the frequency in the range of from 2Mhz to 50 MHz.

6. The user can specify the number of cycles to step the clock. Single stepping, multi-cycle stepping, and free run mode are all supported.

7. Software trigger points have been implemented to check for a given condition each time the clock stops.

2.2. Integration of Software and Hardware Tools

The Software Side: Typically the application interface(API) between a reconfigurable computing application and the FPGA co-processor board consists of a library of routines to perform all of the necessary operations to setup, communicate with and execute a specialized circuit mapped to one or more FPGA chips on the board. When an application is being debugged we may want to invoke the API calls as usual, or we may instead want to invoke special debug versions of the routines which allow us to interactively access the Co-Processor. To start execution of a design on the WildForce board, the application incorporates a call to `WF4_ClkFreeRun()`, this call can be replaced by code such as shown in Figure 3. When the `debug_fpga` flag is set to `true` the GUI for the in-situ debugging tool described above in Section 2.1 is invoked. This approach works well with proprietary software debugging environments in which it is not possible to incorporate the FPGA debugging tool into the debugger directly. The user can simply set a break point some where prior to the `WF4_ClkFreeRun()` call, set the `debug_fpga` variable to true within the software debugger and then continue execution. When the call is reached, the In Situ Debugging tool will be invoked in its place.

There are other API calls which may require interactive debugging. For example, an application may be feeding the circuit data via a fifo between the host bus and FPGA chips. For the WildForce board this is done with a call to `WF_FifoWrite()`. The user may want to query the state of the FPGA chip before and after some data is sent as well as

Type of Simulation	Run Time Minutes (Hours)
Functional	40.5 (.675)
Functional with board model	516 (8.6)
Timing level	46.5 (.775)

Table 1. ModelSim simulation times for a synchronous counter run for 4,194,303 clock cycles.

perform clock stepping etc. All such calls must be addressed to provide a comprehensive solution.

The Hardware Side: FPGA Readback allows the user to observe the hardware state after execution of a given clock cycle. This provides similar information as that provided by a functional simulation. However, traditional hardware simulation is still a useful step in the test and verification process. Commercial hardware simulation environments are backed by years of development and hence have a robust set of features for circuit debugging. More importantly they provide the ability to explore the timing characteristics of a circuit so that a user can find, for example, a fault caused by a hold time violation or other timing related problem. The main drawback is that simulation is expensive, see Table 1 for some simulation times of a simple counter circuit.

With an integrated environment the user can localize an error quickly using in-situ debugging. The state of the circuit can be readback just prior to the occurrence of a fault and those state values can be used to *fast forward* the simulation to the cycle of interest. The simulation tools can then be used to explore the circuit behavior in detail.

Developing this capability has one major challenge which is to map the physical addresses in the FPGA to the corresponding logical elements in the VHDL description of the circuit. The mapping is specified in a file (`.ll` extension) generated by the Xilinx place and route tools. However, interpreting the `.ll` file is complicated by the fact that net names may not be preserved through the synthesis or placement and routing steps. Researchers from Brigham Young University discuss the problem in [3]. Additionally, for timing level simulation, the Xilinx tools produce back annotated VHDL containing information about the FPGA timing. This *new* VHDL design, exposes the user to much more detail than the original circuit description, potentially obscuring the original design hierarchy. For example, the implementation of the counter circuit which used Xilinx core logic, is now exposed to the user. New state elements and nets are introduced and the names for the original elements that are preserved have been mangled to some degree [4].

Signal values can be injected into a simulation via force commands, via ModelSims's Tcl/Tk based user interface, or by using ModelSim's Foreign Language Interface(FLI). Due to the large number of signals for timing level simulation, there is a need to automate this process.

2.3. Current Status

Our Hardware/Software co-debugging environment currently provides support for the Annapolis Microsystem Wildforce platform (See [11]). We have implemented a GUI based on TCL/TK which supports single clock stepping, multiple clock stepping, reading of off chip memory, basic trigger points, and full FPGA readback of chip state. We have use this GUI with a simple application described below. The C++ code for the application was manually instrumented to invoke the GUI instead of calling the WildForce API routine for starting the clock. We are developing the interface between our GUI and the ModelSim simulation tools. Signal values have been manually injected into a simulation using force commands. We have also experimented with driving signal values via ModelSim's Foreign Language Interface(FLI).

3. Debugging of a Simple Application

We illustrate the convenience of the Co-debugger with a simple example. Consider an application which tests a memory chip on the FPGA co-processor board. The FPGA design consists of a 24 bit counter which cycles through memory addresses. A test pattern is written into each memory location which can then be read and verified within the host code for the application. Assume the following faulty result. The test is run on each of the memory chips on the Wildforce board and the program reports that the memory locations for addresses in the range of 0x400000-0xFFFFFF have erroneous data values which do not match the test pattern.

The following explanations give possible scenarios which could cause such a fault. To find the bug the user has to investigate each option and determine either that it is, or is not, causing the problem.

1. All of the memory chips are bad.
2. The test pattern data has not been properly written to memory locations within the given address range.
3. The data in the given address range has not been read back correctly in the host program.
4. The code to compare the test pattern with the value read from the memory contains errors.

Senario 1 is unlikely. If the memory chips are in sockets, it can easily be checked by replacing the chips with know good

ones. Reading of the external memory and comparison with the test pattern are part of the host program thus Scenarios 3 and 4 can be investigated using standard software debugging procedures. However, scenario 2 requires investigation of the FPGA circuit with either the in-situ debugging interface, circuit simulation or a combination of the two.

To run the counter out to address 0x3FFFFF, just prior to the start of faulty operation, requires $2^{22} - 1 = 4,194,303$ clock cycles. Simulation times for a 24 bit counter using Model Technologies ModelSim tool are listed in Table 1. Simulation out to cycle 4,194,303 of just the counter design takes approximately 3/4 of an hour. Since the FPGA chip in which the counter circuit will operate is part of a co-processor board, the user may want to simulate with a full model which includes the entire co-processor. Using the board model provided by AMS, such a simulation will require more than 8 hours to run out to the cycle of interest.

Using the In-Situ interface, the FPGA design can be setup to free run for 4,194,303 clock cycles. For a design clocked at 40ns, running out to cycle 4,194,303 will take about .17 seconds. The overhead of stepping the clock, performing FPGA readback and looking up a signal value has been measured at less than a second. So the user can fast forward to cycle 4,194,303 in just over 1 second. From this point on the clock can be single stepped. The addresses can be verified via the signal readback feature and the memory contents can also be queried as the clock is stepped. If this information is not sufficient for finding the fault then simulation can be invoked starting at clock cycle 4,194,303.

4. Related Projects

The issues addressed by this work are well known and many aspects of this work have been addressed by others. We describe some of the most relevant work here.

Reducing the Cost of Debugging Researchers are developing promising techniques for reducing the synthesis and place and route times with incremental place and route [12], and fine grain and incremental synthesis [2]. These advances will help reduce the overall time for a debug cycle. In addition, cycle based simulators have achieved about 10 times speedup over event driven simulation [9] but they still cannot match the increased speed and capacity of simulation on actual hardware. It has been shown that when a hardware design is emulated on a network of FPGA it can achieve a speedup of 10,000 times over software simulation [10]. This demonstrates the potential performance gains of in-situ debugging.

There are other in-situ debugging tools, for example the InnerView Hardware Debugger from the Virtual Wires group at MIT [4] and the ChipScope tool from Xilinx [1]. Both are stand alone utilities, while ours is designed to be integrated with software and hardware debugging environments.

Other researchers and developers are also investigating co-verification environments for Reconfigurable Computing. Brad Hutchins, et al. have developed JHDL, a JAVA based hardware design environment for FPGA design [5], [6]. Like our approach, it also supports both in-situ and simulation based debugging. We Target C/C++ applications, with FPGA designs in VHDL and integrate existing hardware and software design tools to support existing applications and to provide familiar tools to developers. JoonSeoYim et.al. have also proposed a uniform approach for co-simulation using the RTC C model for Hardware design [13]. They show how large designs can be verified quickly with their methodology. One drawback of there approach is that timing accurate simulation is not provided so timing problems may go undiscovered.

Commercial products such as Seamless CVE^{TM} from Mentor Graphics Corporation for hardware/Sofware co-verification are also available [7]. We are not aware of any that provide a complete solution with support for linking software debugging, in-situ FPGA debugging and hardware simulation. Seamless CVE^{TM}, for example, does not have the capability to accelerate simulation via FPGA.

5. Conclusion

We have presented a debugging environment for use with FPGA accelerated applications which supports co-validation and co-testing of the software and hardware portions of the application. We have shown that this environment has the potential to reduce application debug time for a simple applications from hours to just a few minutes. Such improvements are potentially much greater for applications with more complex FPGA designs since the hardware simulation time for a large design is much longer than simulation of a simple design.

5.1. Extending the Co-Debugger

There are many ways in which the Debugging environment can be extended to make it useful for a wider range of applications. Its efficiency can be improved for applications with very large FPGA designs by supporting simulation of subciruits of a design. Input to the subcircuit can be traced and recorded to inject into the simulation program. In addition it can be extended to support the debugging of asynchronously triggered applications such as a communications application in which work is performed in response to an externally triggered asynchronous event.

The environment has to be extended to support both a trace buffer and real-time trigger points in order to be used with asynchronous applications. Trace buffering can be

added as part of the circuit design but is limited by the amount of memory available. Chips such as the Xilinx Virtex family can potentially use the FPGA block RAM for storage of data traces. Trigger points must be implemented in hardware so that the comparison can be performed every clock cycle. For chips which support partial reconfiguration, debug logic can be mapped to a reconfigurable 'block' of CLBs, thus the trigger events can be changed interactively during a debug session and the number of CLBs required for the trigger logic can be kept small.

6. Acknowledgements

We would like to thank Stephen Carl and Bhavin Juthani for early work on this project. We would also like to thank Jack Jean, Paul Rudolph and Dan Rudolf for the many discussion which contributed to this work. This research has been supported in part by an Ohio Board of Regents Research Challenge Grant.

References

[1] ChipScope ILA^{TM} home page. http://www.xilinx.com/products/software/chipscope/index.htm.

[2] W. J. Fang, A. C.-H. Wu, and T. Y. Yen. A real-time rtl engineering change method supporting on-line debugging for logic-emulation applications. In *Proceeding of 34th Design Automation Conference*, pages 101–106, June 1997. Anheim.

[3] P. Graham, B. L. Hutchings, and B. Nelson. Improving the FPGA design process through determining and applying logicl-to-physical design mappings. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines.*, April 2000.

[4] S. Hanono. Innerview hardware debugger: A logic analysis tool for the virtual wires emulation system. Master's thesis, Massachusetts Institute of Technology, February 1995.

[5] B. L. Hutchings, P. Bellows, J. Hawkins, S. Hemmert, B. Nelson, and M. Rytting. A CAD suite for high performance FPGA design. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines.*, pages 12–23, April 1999.

[6] B. L. Hutchings and B. E. Nelson. Using general-purpose programming languages for fpga design. In *Proceeding of 37th Design Automation Conference*, pages 561–566, June 2000. Los Angeles.

[7] R. Klein and R. Nelson. Seamles CVE^{TM}: Hardware/software co-verification technology. Technical publication, Mentor Graphics Corportation. http://www.mentor.com/seamless/.

[8] Model technology's modelsim home page. http://www.model.com/.

[9] Quickturn home page. http://www.quickturn.com/.

[10] J. A. Rowson. Hardware/software co-simulation. In *Proceeding of 33th Design Automation Conference*, 1996.

[11] Wildforce users manual.

[12] Using xilinx and synplify for incremental designing (eco). Technical Report XAP164, Xilinx, August 1999.

[13] J. S. Yim, Y. H. Hwang, C. J. Park, H. Choi, W. S. Yang, H. S. Oh, I. C. Park, and C. M. Kyung. A c-based rtl design verification methodology for complex microprocessor. In *Proceeding of 34th Design Automation Conference*, pages 83–88, June 1997. Anheim.