VirtexDS: A Virtex Device Simulator

Scott P. McMillan, Brandon J. Blodget, and Steven A. Guccione

Xilinx Inc., 2100 Logic Drive, San Jose, California 95124

ABSTRACT

Until recently FPGAs have been used almost exclusively to implement static circuits. Because FPGAs can be reprogrammed at any time, even in-system at run-time, interest in exploiting this mode of operation has steadily increased. One barrier to widespread use of Run-Time Reconfiguration (RTR) has been the lack of design tools. While tools such as *JBits* have begun to provide basic support for design entry, traditional verification tools such as simulators have been lacking. This paper discusses *VirtexDS*, a device level simulator for the Xilinx Virtex$^{(tm)}$ series. The approach taken by *VirtexDS* is to simulate at the device level, providing an interface which operates much like actual hardware. This approach not only supports simulation for run-time reconfiguration, but also interfaces easily to existing tools. In addition, this low-level simulation approach can provide higher performance than higher-level approaches to simulation.

1. INTRODUCTION

Run-Time Reconfiguration (RTR) tool flows have recently become available with the advent of Application Programming Interfaces (APIs) like *JBits*$^{(tm)}$.[1] Tools such as *JBits* allow applications to modify and download configuration bitstreams on the fly, permitting a new level of flexibility in hardware design.

Unfortunately, existing circuit simulators are coupled to schematic and HDL design tools and provide no support for RTR. To address this gap in existing tools the *Virtex Device Simulator (VirtexDS)* has been designed and implemented as part of the *JBits* tool suite. VirtexDS operates at the device level and provides a software model of the entire Virtex family of FPGAs. In addition to providing support for RTR, this device level simulation approach has other benefits.

First, because the actual FPGA device is being simulated, this approach provides a level of simulation accuracy difficult to attain in other implementations. A device simulator also provides a “safe” environment for testing and debugging of RTR applications. Illegal configurations that would damage actual hardware can be caught and identified. Lastly, because the simulator uses only configuration bitstream data as input, designs are optimized and pre-packed into look-up tables (LUTs). This provides a performance boost which makes *VirtexDS* a high-performance alternative to traditional simulation.

At the system level, the device simulator interface is identical to that of actual hardware. This permits existing *JBits* tools and applications, including the *BoardScope*$^{(tm)}$ debug tool,[2] to interface directly to the simulator with no modifications. This unified hardware / simulator interface simplifies user interaction and provides a powerful debug and test environment.

In Section 2 and 3 we will provide background information on *JBits* and traditional optimization techniques using high level simulators. In Section 4 we will discuss the device simulator approach and talk about its advantages over high level simulation in addition to how optimization techniques used for high level simulation can also be used for device level. Finally in Section 5 we will give information pertaining to our implementation of the device level simulation and we will discuss effects of FPGA architecture designs and conclusions in Sections 6 and 7.

Author contact information:
Scott.McMillan@xilinx.com
Brandon.Blodget@xilinx.com
Steve.Guccione@xilinx.com

2. *JBITS* BACKGROUND

JBits is a set of Java$^{(tm)}$ classes which provide an Application Program Interface (API) into the configuration bitstream for devices in the Xilinx Virtex family. This interface permits all configurable resources in the device to be individually programmed directly under software control. This provides software support for a set of new capabilities previously unavailable 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 API. This provides a simple alternative to traditional design tools.

Figure 1 provides a general overview of all the different components in the *JBits* tool suite. The components shown include the RTPCore library of predesigned cores, the *JRoute* automatic router API, the *BoardScope* debug tool, and the *JBits* configuration bitstream interface. The *XHWIF* Xilinx hardware interface API shown at the bottom of Figure 1 is used to access both actual Virtex hardware and the device level simulator.

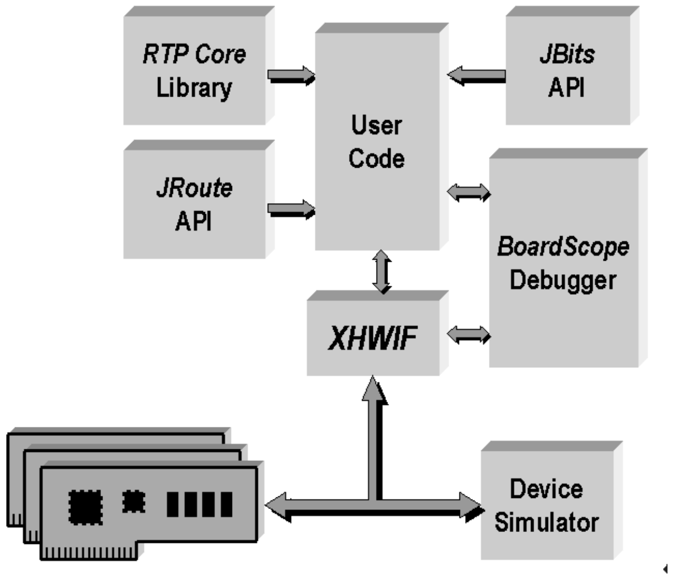

Figure 1. JBits Overview Diagram.

3. TRADITIONAL HIGH-LEVEL SIMULATORS

Currently, most commercially available circuit simulation environments are closely tied to circuit development environments. In general, these tools provide simulations for designs entered using traditional schematic capture or Hardware Description Language (HDL) input. Because these input specifications only support fixed circuits, the simulators used with these design entry tools as they exist today are generally not suitable for modeling RTR.

Figure 2 shows an overview of how traditional HDL type high level simulators fit into the design flow. There are many vendor supplied design entry tools and many vendor supplied simulators to choose from. This increases the complexity of the M1/M2 Place and Route software, for it needs to communicate with many different vendor tools. In the traditional high-level simulator flow all of the simulation takes place up-stream of bitstream generation. If errors are introduced during the generation of the bitstream, these errors can not be caught by the simulator. Also since traditional simulators are designed to simulate static circuits, RTR type designs can not be simulated. The inability to simulate RTR designs provided the original impetus for designing the *VirtexDS* device level simulator.

Circuit simulators can be grouped into two general classes: event driven and cycle based. Event driven simulators are more common and provide circuit timing and delay information. Whenever a signal state changes, an internal *event* is generated. Due to circuit delays, this change of state event will take place at some point in the future. The event is typically placed in an *event queue* and sorted by time. Events are removed from the queues in order and processed. Typically, processing propagates

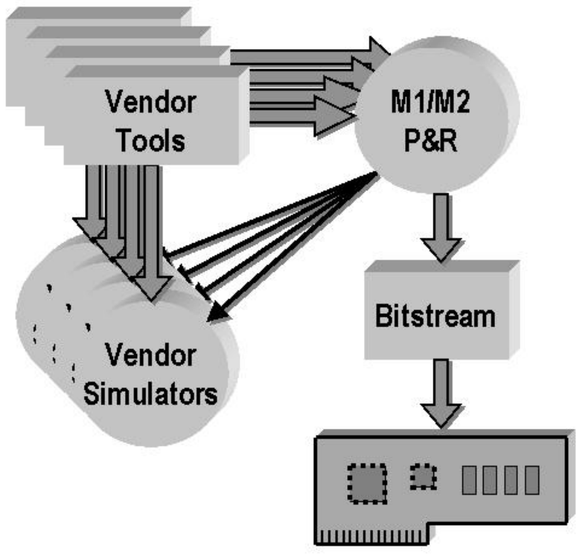

Figure 2. Traditional High-Level Simulator Overview.

the event through the circuit and generates additional events which are added to the queue. This approach provides accurate timing information, but can be somewhat slow.

By contrast, cycle based simulation was designed for speed but only provides a functional simulation, not timing behavior. In a cycle-based simulator, asynchronous portions of a circuit are simulated until some state element, typically a flip-flop, is encountered. On a clock signal, the next-state values generated by the circuit are propagated to the output of the state elements. These new values are then used to produce the subsequent set of next-state values for the state elements. While this approach tends to be faster than event-based simulators, timing behavior is not taken into account and asynchronous designs cannot be simulated at all.

In addition, circuit simulators generally provide more information than the typical one / zero state found in actual hardware. Most modern simulators use 9 state logic from the *IEEE 1164*[3] standard. This has been popularized with its widespread acceptance as a standard VHDL library component. In addition, strength modifiers added to these logic values create a large number of possible states and complicated sets of resolution functions required during simulation. These multiple states and resolution functions can considerably complicate the simulator code and reduce performance. However, they are required to accurately represent real world circuit possibilities and unknown initializations.

For performance reasons, there has been a resurgence of simpler two state simulators[4].[5] This is a natural choice for *VirtexDS*, since the FPGA device always initializes to a known state and changes to circuit parameters such as drive strength are not available to designers.

4. AN FPGA DEVICE LEVEL SIMULATOR

Device level simulation is a new approach to FPGA design simulation and verification. This type of simulation not only provides the ability to simulate RTR designs in a safe environment, but has other inherent advantages over other simulation approaches. Figure 3 shows a high level overview of device level simulation. Device level simulation can take direct place of the actual hardware device and works directly from the configuration bitstream. An added benefit of this approach is that the simulator also becomes tool independent. Any design tool which generates a configuration bitstream for a Virtex device can be simulated with this approach.

The approach to simulation is relatively simple. Logic utilization and connectivity information is extracted from the configuration bitstream using *JBits*. This information is all that is necessary to accurately simulate a design. FPGA devices have the additional benefit that the simulated structures are regular, simple, and small in number. In general, the simulation model consists of interconnected 4-input look-up tables (LUTs) and flip-flops. These types of structures greatly simplify the simulator design, initialization, and memory requirements.

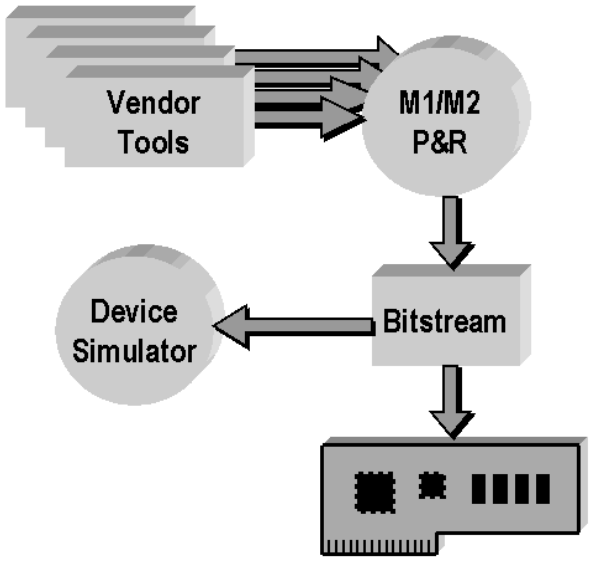

Figure 3. Device Level Simulator Overview.

In addition, the signals in an FPGA device all have a known initialization state and drive strengths are uniform. This eliminates the need for the multiple states of other simulators. *VirtexDS* uses two states: one and zero. This not only simplifies the design and increases performance, but it give a more uniform view of the design. The output of *VirtexDS* should be identical to that of the actual FPGA hardware.

Finally, the basic structure of an FPGA, the Look Up Table (LUT), provides for additional speed and efficiency optimizations over high level simulators. LUTs represent multi-gate functions, providing a condensed version of the original gate level design. This "packed" form of the gate level information generates less overhead during simulation. For example, in an event driven simulator, 20 individual gates will generate 20 times the number of events as the same circuit packed into a LUT. Each of these events will also carry timing information which does not correspond to any physical feature of the device. The gates do not actually exist in the final circuit; they have already been packed into LUTs.

In the *VirtexDS* implementation, the interface used is exactly the same interface used to talk with Virtex hardware devices. As a result, final state information is the only information of interest. Intermediate state information can safely be ignored. This essentially produces a functional simulator which can ignore signal glitches and intermediate states while still providing the capability to detect signal timing violations.

If required, a device level simulator can also provide a higher level of timing fidelity than can be obtained using high level back annotated simulations. A static timing analysis can be performed in addition to the more in depth event driven simulation that displays timing information and every glitch in a signal. All of this can be achieved directly with the device simulator rather than using different tools to annotate and simulate timing information. The timing information for the device level simulator will also be based on actual routing delays of the final circuit implementation rather than hypothetical timing estimates.

5. THE *VIRTEXDS* IMPLEMENTATION

The primary goal in implementing the *VirtexDS* device simulator was to prove some of the concepts of device level FPGA simulation. Since many of the ideas were untried, a path that would consume minimal engineering resources was charted for the first implementation. The goal was to keep the code size as small and simple as possible. Performance was always secondary to producing a working prototype.

Even with these goals, the implementation of the *VirtexDS* device simulator was surprisingly fast. One reason for the small size and rapid development was that existing software and specifications from the *JBits* tool suite were heavily leveraged.

Of course, all of the configuration bitstream information was decoded and interpreted using the *JBits* API. But other parts of *JBits* were also used. For instance, rather than define a new interface to the simulator, the decision was made early to provide an *XHWIF* portable hardware interface to the simulator. This would allow standard tools such as the *BoardScope* graphical debugger to provide a solid, familiar user interface. In addition, the use of *XHWIF* also allowed existing *JBits* applications to use the simulator directly without rework or recompilation.

As an example, one application experimenting with *Evolvable Hardware*[6] was able to make excellent use of this *XHWIF* simulator interface. Originally this application ran on a Virtex-based PCI board. Runs of this application would often take days, tying up valuable resources. Once the simulator became functional, multiple runs of this application were spawned off in parallel on local workstations. Not only were results generated much faster, but the hardware was freed up for other developers.

The next piece of *JBits* code to be leveraged in this effort was the database for *JRoute*, the run-time router API.[7] This database provided connectivity information for the Virtex family of devices. This was used to implement the *tracer* which was used to build up the interconnection graph of the bitstream. This tracing was done only when new bitstream information was acquired, either through loading a completely new design or via reconfiguration. This process, however, turned out to be relatively time-consuming on start-up. A variant of the original tracer was eventually implemented which uses a greedy algorithm to trace only objects which generate events in the simulator. This amortizes the costs of tracing routes over time instead of forcing the user to wait for tracing at initialization.

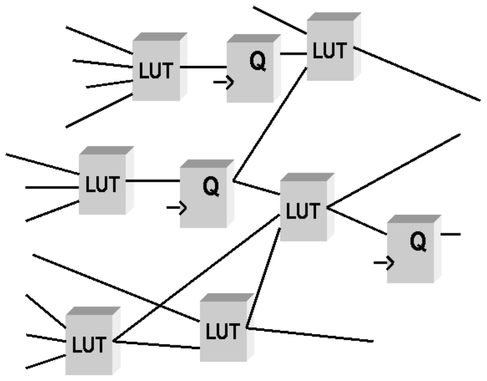

Figure 4. A simplified extracted circuit graph.

As the routing interconnecting individual LUTs and flip flops is traced, an internal representation of the circuit is built. This representation is essentially a graph containing interconnected LUTs and flip flops. A simplified diagram of an extracted circuit is shown in Figure 4. The delay of each route is calculated once and stored with the associated interconnection arc. This provides very accurate timing information without having to actually simulate the details of the FPGA routing architecture. This dynamically constructed graph is the FPGA circuit representation used to drive the simulation.

Table 1. VirtexDS Numbers

Description	Number
Lines of Code	2351
Design Time	3 Man Months
Events Per Second	126,000 to 292,000
Tracer Sinks/Second	2,800 to 17,000
Memory Requirements	22 MB on XCV800 (84% LUT’s, 94% FF’s)
Test Machine	450 MHZ Pentium II, 384 MB RAM
Java VM	Sun’s JDK1.2.2. Note: using Sun’s HotSpot VM improves the above numbers on average by 75%

The reporting of simulation results is also simplified. Because the *XHWIF* interface only requires the end result of the simulation cycles, design modifications to the event queue were made. The *VirtexDS* event queue is usually not sorted. This may seem counterintuitive, but results in simplification of the code in addition to reductions in computation. By not sorting the queue, not only can the expensive sorting computation be skipped, but the latest events can be processed first, ignoring the “glitches” generated by earlier events on a signal. Tests showed that approximately 37% of the events generated were able to be ignored in this manner.

Finally, in order to deal with asynchronous circuit behavior, an event queue sorting routine is provided as an option. This is enabled only when asynchronous clocks exist in an FPGA bitstream. This will reduce the speed and efficiency of the simulator, but will permit these types of circuits to be accurately simulated.

It should be noted that while cycle based simulation seems appropriate for this type of model, a variant on the event-based model is used in *VirtexDS*. This approach is used not only to support simulation of asynchronous circuits, but also to support simulation with timing data. Because the circuit being simulated represents the physical realization in the FPGA, more accurate circuit timing simulation can be performed using *VirtexDS*.

Table 1 provides some quantifiable data relating to the *VirtexDS* device simulator. These numbers varied widely in some cases as a result of the number of actual events processed per cycle. Since there exists overhead with calling the event queue, the more events processed per cycle, the better the average rate. Similar behavior exists with the tracer where sink to source ratios have a large effect on the average tracer sinks per second rates. Overall, the memory consumed by *VirtexDS* and the speed of simulation are quite notable. Comparable commercial simulators use approximately an order of magnitude more memory and can be as much as two orders of magnitude slower.[8]

6. FUTURE WORK

The first version of *VirtexDS* turned out to be smaller and faster than even the most optimistic estimates. It has quickly become the preferred vehicle for development of designs and has all but eliminated our reliance on physical hardware. While the results are encouraging, development on *VirtexDS* is continuing.

The first planned enhancement to *VirtexDS* is to support the entire Virtex device. Currently the Virtex block RAM and IOBs are not supported. In addition, some method of driving the external IO ports in the device simulator would be useful in the testing of complete designs.

The current simulator is event based, but actual numbers for the various speed grades of Virtex devices are not available. It is expected that the timing data from the mainstream M1 / M2 placement and routing tools can be leveraged to supply this data to *VirtexDS*. As with other aspects of *VirtexDS*, relatively little timing data will be required. We predict that values for LUT delay, flip flop delay and delays for single, hex and long lines will be the bulk of the data required. Note that the current version of the simulator assumes unit delay. The addition of these values will not affect performance.

Finally, we would like to build device simulators for other Xilinx devices. In particular, producing such tools before silicon is available for new families would help in the development and debug of not only the mainstream tools, but also of intellectual property and designs. Ultimately, we would like to use *VirtexDS* as a tool for experimenting with FPGA architectures. Architectural variants, or even completely new architectures could be implemented in the style of *VirtexDS* and tested and exercised without having to resort to large, expensive design tools or produce silicon.

7. CONCLUSIONS

VirtexDS provides simulation support for run-time reconfiguration, which was actually the ultimate goal of this work. In addition, device level simulation of FPGAs provides many inherent advantages over traditional simulation. First, device simulation offers very high performance over existing techniques. This performance is achieved through simulating a simple, regular array or logic already packed into LUTs and the ability to perform accurate simulations using only two states.

Somewhat surprisingly, accuracy is also enhanced. The circuit being simulated is the actual FPGA implementation, not some gate-level representation. This allows more realistic and accurate timing information to be produced by the simulator. Finally, the two state nature of the device simulator provides data that should be identical that that of actual hardware. Complex set up and initialization of the simulator are not required; they are a built in feature of the FPGA device.

Aside from the inherent advantages of this approach over traditional simulation, *VirtexDS* was able to provide additional benefits. First, leveraging existing code from the *JBits* toolset greatly simplified the design and dramatically reduced the code size. All devices in the Virtex family are supported in under 2500 lines of Java code. And the use of existing interfaces, particularly *XHWIF*, allowed existing user interfaces to be used and permitted existing applications to run unmodified and without recompilation on *VirtexDS*.

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," 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.
2. 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., pp. 239–246, SPIE – The International Society for Optical Engineering, (Bellingham, WA), November 1998.
3. The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017 USA, *IEEE 1164-1993. IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164)*, 1993.
4. L. Bening, "A two-state methodology for RTL logic simulation," in *Proceedings of the 36th ACM/IEEE conference on Design automation conference*, pp. 672–677, IEEE Computer Society Press, June 1999.
5. H. K. Chaudhry, P. Eichenberger, and D. R. Chowdhury, "Mixed 2-4 state simulation with vcs," http://www.synopsys.com/products/simulation/ivcpaper.html, Synopsys, Inc., April 1998.
6. D. Levi and S. A. Guccione, "GeneticFPGA: Evolving stable circuits on mainstream FPGAs," in *Proceedings of the First NASA/DOD Workshop on Evolvable Hardware*, A. Stoica, D. Keymeulen, and J. Lohn, eds., pp. 12–24, IEEE Computer Society Press, (Los Alamitos, CA), July 1999.
7. E. Keller, "JRoute: A run-time routing API for FPGA hardware," in *Parallel and Distributed Processing*, J. Romlin et al., ed., pp. 874–881, Springer-Verlag, Berlin, May 2000. 7th Reconfigurable Architectures Workshop (RAW), part of the Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS) Workshops 2000 held in Cancun, Mexico May 1-5, 2000. Published in the series Lecture Notes in Computer Science 1800.
8. L. Geppert, "Electronic design automation," *IEEE Spectrum* **37**, pp. 70–74, January 2000.
9. Xilinx, Inc., *Xilinx Data Book*, 2000.
10. M. J. S. Smith, *Application-Specific Integrated Circuits*, VLSI Design Series, Addison-Wesley Publishing Company, 1997.
11. S. Yalamanchili, *VHDL: From Simulation to Synthesis*, Xilinx Design Series, Prentice-Hall, Inc., July 2000.

Ã'Ã˜.) can be inferred in most. cases arising in practice [4], allowing us here to write the more concise. Ã–Ã– ÃÂºÃŠ Ãš Ã–Ã— Â´ Ã–Ã–Âµ. . 2.4 Dening parameterized types. The previous examples have shown the use of parameterized types and methods, though not their declaration.Â Ã Ã'ÂºÃ– Ã–Ã˜. ÂºÃ' Ã˜ Ã" Ã"Ã™ Ã Ã—Ã˜ Ã˜ ÃšÃ" Ã… Ã'Â´Âµ ÃŸ Âº Ã'Ã˜Ã–ÃÃ"Ã" Ã'Ã˜ ÂºÃ' ÃœÃ—Ã˜ Â¿ ÂºÃÃ" ÃÃ— Â´ Ã Ã—Ã— Ã‹Ã˜ Âµ Ã' Ã›Ã" ÃšÃ" Ã‹Ã˜ Âº Ã˜Ã"Ã–Â´Âµ Ã—Ã˜ÃÃ" ÂºÂ¼ Ã ÃÃ" ÂºÂ¼ ÃÂº Â½ Ã"Ãœ Ã‹ÃÃ—Ã˜ Ã'ÂºÃÃ'Ã˜Â¿Â¾ ÃÃ'Ã—Ã˜ Ã' ÃšÃ" Ã‹Ã˜ ÃˆÃ™Ã— Â´ Ã Ã—Ã—. Ã‹ÃÃ—Ã˜ Ã'ÂºÃ‡ Ã ÃÃ" ÂºÂ¼ ÃÃ Ã'Ã—Ã˜ Ã' Ã Ã—Ã— Ã‹ÃÃ—Ã˜ Ã'ÂºÃ‡ Ã˜ Ã‹Ã˜ ÃˆÃ"Ã"Â´Âµ Ã™Ã' Ã"Ãœ Ã‹ÃÃ—Ã˜ Ã'ÂºÃÃ'Ã˜Â¿Â¾ Ã Ã'Âº ÃÂº Â½ Ã• ÃÃ ÃšÃ" Ã‹ÃÃ—Ã˜ Ã'Âº Ã"Ã'Ã—Ã"Ã ÃÃ– Ã˜ Ã„ Ã' Â´ Ã"Ã"ÃÂµ Ã–Ã˜. Ã˜Âµ. Generic stack. Ã Ãš Ã"Ã"ÃÃÃ'Ã"Ã' ÃÃ—Âº. Âº Ã‹ÃŒÃŠ ÃˆÃ‡Ã„ Ã… ÃŠÃ ÃŒÃÃ‡Ã† Ã…Ã‡ Ã„ÃÃ†. Ã˜ Ã¶Ã³Ã± Ã ´Ã³Ã°Ã½Â¹Âº Ã²Ã½Ã° Ã¸ Ã¸ Ã«Ã¬Ãª Ã— Ã±Ã¹Ã° Â¹. Ã˜ Ã³Ã² Â´Ã¬ Ã½Ã±Ã³Ã¹Ã¶Â¸ Â½ Âµ Ã— Ã¹Ã— Ã¸Ã³ Ã°Ã°Ã¹Ã—Ã¸Ã¶ Ã¸ Ã±Ã³ Ã°. Ã'Ã¸ Â¬ Ã¸ Ã³Ã²Âº Ã¬ Ã³Ã¹Ã¸Ã´Ã¹Ã¸Ã— Ã¶ Ã¶ Ã¸Ã³Ã¶ Ã¸ Ã±Ã´ Ã¶ Â¹. ÃÃ°Ã¹Ã° Ã¶ Ã¹Ã¸Ã³Ã± Ã¸ Ã¥Ã³ Ã°Ã— Ã³Ã¶ Ã« Ã±Ã¹Ã° Ã¸ Ã³Ã² Ã³ Ã³Ã±Ã´Ã¶ Ã—Ã— Ã° Ã°Ã¹ Ã½Â¹. Ã' Ã' Ã‹ÃÃ—Ã˜ Ã'Ã— is an original research work carried out by Ã€ Ã' Ã'Ã— Ã™ Ã– Ã› Ã under. my supervision, and that this work has not been submitted elsewhere for a degree.Â ÂºÂ½ÂºÂ¾ Ã§Ã² Ãª Ã—Ã¸Ã³Ã¶ Ã² Ã«Ã´ Ã¸ Ã° Ã¥Ã³Ã± Ã²Ã¸Ã¹Ã± Ãª Ã—Ã¸Ã¶ Ã¹Ã¸ Ã³Ã² Ã¹Ã¶ Ã² Ã¡Ã²Ã¸ Ã¶Ã´ Ã¶Ã¸ Ã° Ã¡Ã²Ã¸ Ã¶ Ã¸ Ã³Ã²Ã— Ã² Ã¥Ã¹Ã°Ã¸ Ã´ Ã¶Ã¸ Ã° Ã¤ Ã¸Ã¸ Ã— Ã—. From the previous section (Sec. 4.1.1) it is clear that in multiparticle lattice gases spa-tial momentum redistribution during interparticle interactions can be restored only by incorporating interactions among particles occupying dierent lattice sites in their colli-sion rules.