DEEP: An Iterative FPGA-based Many-Core Emulation System for Chip Verification and Architecture Research

Juergen Ributzka, Yuhei Hayashi, Guang R. Gao
University of Delaware
140 Evans Hall
Newark, DE 19716
{ributzka,hayashi,ggao}@capsl.udel.edu

Fei Chen
ARM Inc.
3711 S. Mopac Expressway
Building 1, Suite 400
Austin, TX 78746
fei.chen@arm.com

ABSTRACT

This paper introduces the Delaware Enhanced Emulation Platform (DEEP) - a FPGA-based emulation system for hardware/software co-verification of many-core chip architectures. This platform exhibits the following three characteristics: fast compilation of logic designs, debugging support, and affordability. It is based on a novel iterative emulation methodology for hardware design and verification.

We also conducted a logic design and integration of a new architectural feature that provides Full/Empty bit fine-grain synchronization for the IBM Cyclops-64 many-core architecture and evaluated its performance against existing synchronization constructs.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Measurement techniques

General Terms

Design, Verification, Performance

1. INTRODUCTION

Currently, full-system verification still requires an armada of computers or expensive specialized hardware [1, 4] to achieve reasonable emulation speed. A cluster of computers can be made easily available to a larger group of developers, but the overall emulation speed is still limited. On the other hand, specialized hardware is much faster, but it is a scarce resource. Faster and cheaper hardware emulation and verification systems are needed to mitigate this problem.

The need for better and faster verification frameworks is growing even stronger with the introduction of new execution models for massive many-core designs. These models provide feedback to hardware architects about possible advantageous features. This results in a symbiotic relationship, which requires hardware/software co-development methodologies. These new methodologies have caught the attention of many high profile research institutions, including DARPA, which is funding a project with the main objective to explore these techniques [2]. Development of such techniques would greatly benefit from better, faster and affordable emulation systems.

Among the new many-core designs, we have the IBM Cyclops-64 (C64) many-core architecture. The architecture consists of 160 homogeneous processing elements called Thread Units (TUs). These 160 TUs are connected via a high speed crossbar interconnect and share 4.7 MiB of internal memory. There are no data caches on this architecture - only Instruction-Caches (ICs). Moreover, the architecture has support for hardware based barriers. Finally, all levels of the memory hierarchy are fully software managed and distributed across three different memory spaces: Scratch Pad memory, the Global Interleaved SRAM and the off chip DDR2 memory.

To test the C64 architecture's hardware features and software stack, a custom made FPGA-based emulation system (DEEP) was created. The emulation system for the C64 architecture was specifically designed and built for many-core architecture emulation and verification. Its unique iterative approach allows the emulation of huge many-core systems with a limited set of FPGAs based on a methodology introduced in [6].

The original work behind this paper has covered two distinct topics: many-core emulation methodologies and architectural research. However, due to the space limitation, this short paper will mainly focus on the first topic. For a more complete coverage of both topics please refer to [5].

The remainder of the paper is structured as following: Section 2 explains the emulation system, its emulation methodology, and its debugging features. Section 3 evaluates the implementation of fine-grain synchronization and provides performance results compared to other synchronization constructs. Section 4 gives a overview of related work and Section 5 concludes the paper.

2. DEEP: THE EMULATION SYSTEM

In this section we describe the hardware platform, the emulation methodology and the debugging support of the Delaware Enhanced Emulation Platform (DEEP). DEEP has been developed in order to validate the C64 chip's hardware features and test its software stack. It can be ported to emulate other many-core architectures too. The major objectives of DEEP are to support all design and test stages, to realize good turn-around time for the early stages and high emulation speed for the later stages of development, to do the whole chip emulation as well as to provide an efficient debugging environment.

2.1 DEEP Hardware Platform

The DEEP hardware platform is comprised of a host system and a custom made system with a series of highly connected FPGAs. Figure 1 shows the block diagram of DEEP.

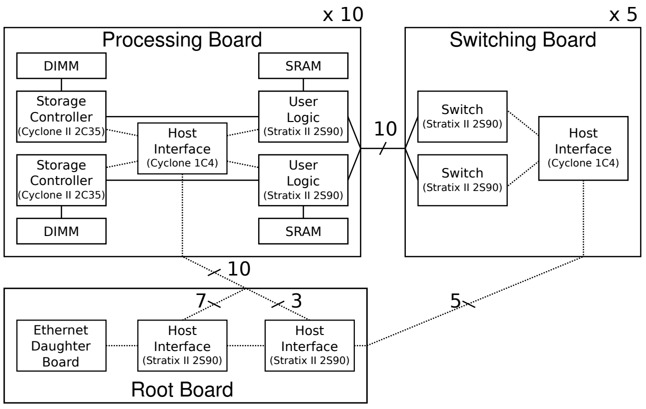
x 10
Processing Board
DIMM
SRAM
Storage Controller (Cyclone II 2C35)
User Logic (Stratix II 2S90)
Host Interface (Cyclone 1C4)
Storage Controller (Cyclone II 2C35)
User Logic (Stratix II 2S90)
DIMM
SRAM
x 5
Switching Board
Switch (Stratix II 2S90)
Switch (Stratix II 2S90)
Host Interface (Cyclone 1C4)
10
10
7
3
5
Ethernet Daughter Board
Host Interface (Stratix II 2S90)
Host Interface (Stratix II 2S90)
Root Board

Figure 1: Block Diagram of DEEP: The figure shows the tree-like connections between the FPGAs and the different board types - root board, processing boards, and switching boards.

16 FPGA boards are plugged into the backplane, which provides not only power, but also the global clock (100 MHz) and interconnection to all FPGA boards. There are three different type of FPGA boards: root board, processing boards, and switching boards. The root board has two Altera Stratix II 2S90 FPGAs and a daughter board for the Ethernet connection to the host system. The Ethernet daughter board has additional logic, which allows the remote programming of all FPGAs in the system via Ethernet. The FPGAs on the root board are used to implement the root node of a tree. The remaining FPGAs in the system are connected in a tree like fashion to the root board. This allows the host system to communicate with all FPGAs. The processing board has five FPGAs. One Cyclone 1C4 FPGA for the tree node, two Stratix II 2S90 FPGAs for the user logic, and two Cyclone II 2C35 FPGAs for interfacing logic to the DIMMs. These additional FPGAs for the memory interface are required to refresh the memory while reprogramming the user logic in the other FPGAs. The switching board has two Stratix II 2S90 FPGAs, which are used to implement the switching logic for the emulation system. The processing boards are connected via the backplane to the switching boards. These connections are used during emulation to pass data between the different processing boards. The tree connection is only used by the host for communication with the FPGAs. Overall, only 20 Stratix II 2S90 FPGAs can be used for emulating user logic.

2.2 DEEP Emulation Methodology

In order to achieve its main objective, DEEP supports two different modes: simulation mode and emulation mode. The simulation mode is a *logic processor* based logic simulation methodology. In this mode, the original logic design is translated into logic programs. This means the logic design, which usually consists of a netlist of gates and memory cells, is mapped to a series of logic instructions. Therefore, the *logic processors* are able to simulate any logic design. These instructions are executed on a large number of *logic processors* on the processing FPGAs. Figure 2 shows the translation of a logic design into a logic program. DEEP can quickly generate logic programs from an original logic design, because only a simple translation from a netlist to an instruction stream is required. No synthesize is required,

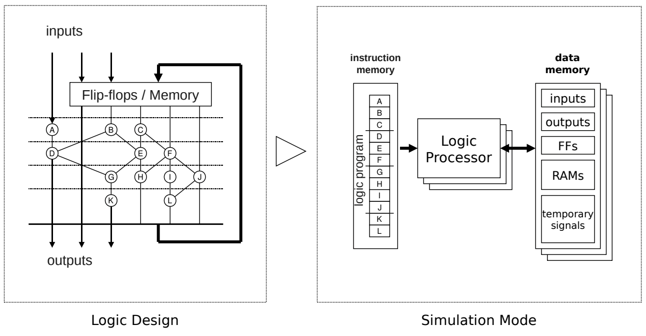

Figure 2: Simulation Mode: User Logic Design to Logic Program mapping. The logic primitives (A-L) shown in the original logic design on the left are translated into instructions for the logic processor shown on the right.

because the *logic processors* do not change. For instance, the C64 combinatorial logic design (around 43 million gates) can be translated into logic programs within two minutes. Logic programs generated from an original design are executed on a huge number of *logic processors* (400 in DEEP). Each processing FPGA has 20 *logic processors*. One instruction queue is shared by all *logic processors* in one FPGA. There are only 20 processing FPGAs, therefore at most 20 different submodules in a logic design can be simulated in this system. If one submodule has more than 20 instances, then multiple processing FPGAs are utilized for it. The simulation mode is also available on a general workstation, so logic simulation can be done anywhere without the DEEP hardware, although the simulation speed is much slower. In case of the C64 design the average simulation speed of the whole chip is around 110 cycles/second. We do not have any comparable numbers for the software version on a workstation, because we never simulated the whole chip in software. We only simulated a subset of the C64 chip (only 10 Thread Units), which is just a 16th of the whole design, with approximately 1 cycle/second.

On the other hand, the emulation mode design is based on an iterative emulation methodology [6]. Since the whole many-core architecture design cannot fit into a single FPGA of DEEP (or any current available FPGA on the market), the architectural design is separated into submodules. Even though each submodule fits into one FPGA, a lot of FPGAs would be required to implement the entire chip in the emulation system. In particular, the C64 chip would require 236 FPGAs to emulate the whole chip logic. Instead of mapping each submodule to a different FPGA, the emulation system adopts an iterative emulation approach (see Figure 3). Combinatorial logic equivalent submodules are implemented on only one (or a few FPGAs), and then iteratively utilized to emulate all instances of the submodule. This emulation methodology drastically reduces the necessary number of FPGAs. Each submodule's FFs and internal RAM blocks are isolated from the original logic design. The content of the FFs and RAMs are independent of each submodule's instance, so they must be stored separately. The emulation system utilizes internal memories for FFs and external memories for RAM blocks, and only the combinatorial logic is implemented in the FPGA. The flow described above is done by the DEEP software automatically for the C64 architecture. The partitioning of the full logic design

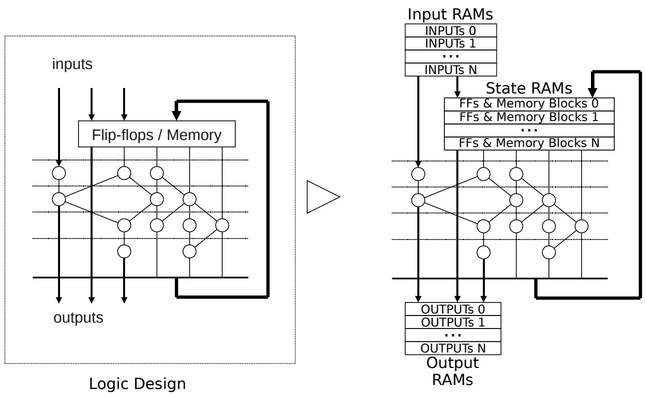

Figure 3: Emulation Mode: User Logic Design to Iterative Emulation mapping. FFs and RAMs are extracted from the original logic design on the left and mapped to instance addressable memory blocks in the FPGA. The combinatorial logic is used iteratively in the FPGA.

is currently done manually and other architectures would require the same manual procedure of partitioning and assigning submodules to FPGAs. Because logic design needs to be synthesized and mapped into FPGAs, it takes much more preparation time than the simulation mode until the logic design is ready to be emulated. Since the submodules can be synthesize in parallel, the whole process takes around 2-3 hours for the C64 chip. However, after the logic design is mapped into the FPGA, it works as real logic on a FPGA, even though it is required to emulate the logic iteratively. In case of the C64 design the average emulation speed of the whole chip is around 80k cycles/second.

2.3 DEEP Debugging Support

At last we discuss the debugging support in DEEP, which is available for simulation mode and emulation mode.

In simulation mode, there are two ways to obtain signals. If inputs, outputs and contents of FFs/RAM blocks of a submodule need to be observed, the DEEP host can directly accesses the external memory, where the target data is stored. For the other signals, additional processing is required, because all intermediate signals are in the local temporary memory of a *logic processor* and may be overwritten. Furthermore, they are also unreachable for the DEEP host. Additional debugging control logic is required to perform the following steps: First, the DEEP host sets a breakpoint in the debugging special-purpose register of the *logic processor*. Second, the *logic processor* starts execution until the program counter reaches the breakpoint. Third, the debugging control unit issues several logic instructions to move the value of the signal to the external memory of the FPGA. Finally, the DEEP host loads the data from the external memory. For this debugging feature, there are 16 special purpose registers available. If more than 16 signals in one submodule are necessary to be observed at the same time, the DEEP host needs to repeat this process for every set of 16 signals. By utilizing both ways to obtain signals, it is possible to achieve 100% signal debugging coverage. Moreover, not only simple signal tracing is possible, but also program tracing is supported, when a processor is simulated.

In emulation mode debugging support is very useful to locate bugs in long running benchmarks. There are again two ways to obtain signals in emulation mode. The first way is the same as for the simulation mode. Input, outputs, FFs, etc of a submodule can be directly accessed by the DEEP host. Unfortunately, in this mode all combinatorial logic is mapped into the FPGA, so it cannot be observed directly. A software simulator on the DEEP host is used in conjunction with the hardware emulator to obtain signals inside combinatorial logic. All the required content from the inputs, FFs and memory blocks of a submodule is obtained by the DEEP host and the resulting combinatorial signal is calculated by the simulator. Even though it is technically not possible to observe all signals inside the FPGA, the missing signal can still be simulated on the DEEP host to obtain full signal debugging coverage. Of course also program tracing is supported in emulation mode if a processor is emulated.

3. CASE STUDY

We used the DEEP platform to enhance the Cyclops-64 architecture with Full/Empty Bit support. In this section we evaluate the performance benefits of our fine-grain synchronization extension. In particular, we took a closer look at a wavefront computation-style program. We created a micro-benchmark, which resembles the data dependencies of a wavefront computation. Since obeying these dependencies are the critical factor of any wavefront computation, this micro-benchmark should be able to give us an estimate on the performance benefits of Full/Empty Bit support versus traditional synchronization constructs.

Due to the dependence of an element on its previously computed neighbors, parallel versions of the wavefront kernel require synchronization constructs to ensure correctness.

We implemented the wavefront computation kernel in four different versions. The different versions are Serial, Barrier, Full/Empty Bit Busy-Wait, and Full/Empty Bit Sleep-Wakeup. The Full/Empty Bit Busy-Wait version constantly polls the memory location until the value has been written. On the other side, the Full/Empty Bit Sleep-Wakeup version only tries once to read the value and then goes to sleep. The producer will wake up the consumer once the value is available. All kernels were completely hand-coded and optimized in assembly to allow for a fair comparison. We run the benchmark on the emulation system for problem sizes starting at 16x16 at increments of 16 up to the maximum supported problem size of 512x512 elements. For each problem size we run the wavefront benchmark with different numbers of threads. Starting from one thread all the way up to 159 threads[1] at increments of one. The numbers collected were measured only for the kernel part of the application and the speedup was calculated based on the results of the serial version.

For the barrier version of the benchmark we achieved a maximal speedup of 24x. Even though the barrier is very efficient, because it has hardware support, the speedup of the application is limited. This is due to the weakest link in the chain, which is the slowest thread. All other threads have to wait for the slowest thread before they can continue doing useful work. Using barriers for these kind of workloads is not necessarily a good choice and dynamic scheduling approaches have achieved better results. We are aware of this, but we chose to demonstrate the barrier implementa-

[1]Only 159 threads can be used, because the OS micro kernel is running on the first thread unit

tion for two important reasons. First, the barrier is a hardware supported synchronization construct and we wanted to compare different hardware supported synchronization constructs. Second, from a programing point of view the barriers seems to be an easy and efficient construct, because the work for each thread is the same. We wanted to show that this thinking cannot be applied anymore to many-core architectures and that congestion, bank conflicts, etc can have unpredictable impacts on a threads execution.The Full/Empty bit versions of the benchmarks achieved much better speedups of 60x and 50x respectively. We took a closer look at both benchmarks by using performance counters. In summary we can say both versions of the Full/Empty bit implementation are not memory bound. The busy-wait version has a synchronization failure rate of 150%. That means every synchronizing load operation has to be repeated 1.5 times in average, because the data had not been written yet by the producer. The sleep-wakeup on the other hand had a failure rate of only 1-2%. Nevertheless, the busy-wait approach still achieved better speedups. The second approach generates less memory operations and also saves power, but the price is a longer synchronization delay, which hinders parallelism and therefore performance.

4. RELATED WORK

There have been many logic verification technologies and products developed in both academia and industry [7, 3]. These technologies have been used to address the many challenges in the logic verification processes.

The iterative emulation methodology, which is adopted in the emulation mode of DEEP, was introduced by Dr. Sakane et al. [6] in 2003. Their emulation system was implemented into one Xilinx Virtex-II FPGA, because of the limited size of the verification target. Moreover, there is no debugging support.

ASIC based logic verification environments and emulation systems have been developed from many Electrical Design Automation (EDA) tool vendors. Mentor Graphics Veloce platform and Cadence Palladimu system are good examples. The Veloce platform [4] is an ASIC based logic verification system developed by Mentor Graphics. This hardware accelerated logic simulation platform utilizes a custom designed emulation chip that contains a programmable logic block for the target logic and a fixed functional block. This fixed functional block handles signal tracing and interconnecting operations. The Palladium platforms [1] from Cadence provide simulation acceleration and in-circuit emulation in a single system.

The RAMP [7] system developed at Berkeley is a FPGA-based many-core emulation platform. This system deploys Xilinx Virtex-II Pro FPGAs on 16-21 BEE2 boards to implement a many-core system composed of 1000 plus cores. The purpose of this project is to explore the architectural design space for future many-core computer architectures and enable early software development and debugging.

A more detailed collection and comparison of related work is presented in [5].

5. CONCLUSIONS

This short paper presented our emulation platform DEEP with its simulation mode, emulation mode, and debugging features. A more detailed explanation of the emulation methodologies and a full coverage of the fine-grain synchronization architecture research can be found in [5].

We also presented a study on how to enhance a many-core chip design with a novel architecture feature using the DEEP framework. In this study, we used the frameworks extensive debugging capabilities to isolate and fix several bugs in our design. Finally, we found that by adding fine-grain synchronization to the C64 design, we can get substantial performance improvements (60x speedup versus 24x speedup) in wavefront like applications.

We also observed a shortcoming of DEEP in the connection between the DEEP host system and the DEEP emulation platform. The current connection uses gigabit Ethernet, which it is unsuitable for tracing hundreds of cores. This requires more research in compact parallel tracing formats and better interconnects.

6. ACKNOWLEDGMENTS

Our utmost respect goes to Monty Denneau for creating such a great architecture. This work would have not been possible without the support by NSF (CNS-0509332, CSR-0720531, CCF-0833166, CCF-0702244), and other government sponsors.

7. REFERENCES

[1] Cadence. Incisive Palladium Series. http://www.cadence.com.

[2] DARPA. Ubiquitous High Performance Computing. https://www.fbo.gov/spg/ODA/DARPA/CMO/DARPA-BAA-10-37/listing.html.

[3] J. Darringer, E. Davidson, D. J. Hathaway, B. Koenemann, M. Lavin, J. K. Morrell, K. Rahmat, W. Roesner, E. Schanzenbach, G. Tellez, and L. Trevillyan. EDA in IBM: Past, Present, and Future. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 22:1476–1497, 2000.

[4] M. Graphics. Veloce SoC Verification System. http://www.mentor.com.

[5] J. Ributzka, Y. Hayashi, F. Chen, and G. Gao. CAPSL Technical Memo 103: DEEP: An Iterative FPGA-based Many-Core Emulation System for Chip Verification and Architecture Research, December 2010.

[6] H. Sakane, L. Yakay, V. Karna, C. Leung, and G. Gao. DIMES: An Iterative Emulation Platform for Multiprocessor-System-on-Chip Designs. In *2003 IEEE International Conference on Field-Programmable Technology (FPT), 2003. Proceedings*, pages 244–251, 2003.

[7] J. Wawrzynek, D. Patterson, M. Oskin, S. Lu, C. Kozyrakis, J. Hoe, D. Chiou, and K. Asanovic. RAMP: Research Accelerator for Multiple Processors. *IEEE Micro*, 27:46–57, 2007.