

# Virtualized Full-System Emulation of Multiprocessors using FPGAs

Eric S. Chung, Eriko Nurvitadhi, James C. Hoe, Babak Falsafi, Ken Mai  
Computer Architecture Lab at Carnegie Mellon (CALCM)  
Carnegie Mellon University, Pittsburgh PA 15213  
{echung, enurvita, jhoe, babak, kenmai}@ece.cmu.edu

## 1 INTRODUCTION

After years of sustained focus on uniprocessor performance, the “power wall” has started the microprocessor hardware and software industry down the multiprocessing (MP) path. This transition raises questions on how to study and design future MP architectures that could incorporate hundreds or even thousands of processors. To compound the problem, innovative MP architectures will require a new, correctly matched software base (OS, compiler, applications) to demonstrate their full potential. However, software researchers neither can wait out the long hardware development cycles nor pursue significant code development on simulators.

In light of this dilemma, FPGAs have attracted the attention of researchers as a fast, flexible and scalable experimental platform for MP architecture studies and software development (e.g., <http://ramp.eecs.berkeley.edu/>). FPGA **emulation**<sup>1</sup> can achieve less than 100x slowdown relative to a real system, a performance deemed acceptable by many software researchers. However, transitioning from the current practice of software simulation into FPGA emulation introduces two major challenges. First, assembling a large-scale (up to 1000-way) multiprocessor prototype, even in FPGAs, is a non-trivial integration effort. Second, full-system emulation—necessary to run operating systems and important commercial applications—requires considerable design knowledge and resources to implement in FPGAs.

Our goal in the PROTOFLEX<sup>2</sup> project is to develop a practical approach to **architectural full-system emulation** of large-scale MP systems. The PROTOFLEX approach centers on the concept of “virtualization” to decouple the capability and complexity of the emulated *target* system from that of the emulation *host*. This paper presents two virtualization techniques that we are investigating. Both techniques enable an emulator to exhibit *more* logical behaviors and resources than what is actually implemented (or even implementable) on the FPGA.

The first technique, **hybrid emulation**, virtualizes the ISA and I/O devices over a combination of hardware emulation and software simulation hosts. In a hybrid emulation, only the frequently exercised portion of a system needs to be implemented in the FPGA, while infrequent, complex behaviors are relegated to software simulation. Executing these dynamically rare

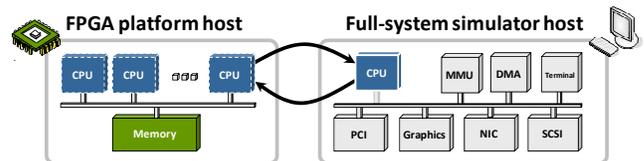
behaviors in software greatly reduces hardware implementation complexity but has little impact on emulation performance.

The second technique, **time-multiplexed emulation**, multiplexes multiple emulated processor instances onto fewer physical instances—providing the appearance of more processors in the system than physically on the FPGA host. This decoupling allows the user to scale the physical implementation only *as needed* to meet a desired emulation performance, independent of the target system size. We further describe the two virtualization techniques in the next two sections.

## 2 HYBRID EMULATION

Only a small subset of total system behaviors actually contributes to the great majority of runtime. However, in a full-system environment, a wide range of supporting ancillary behaviors is required—from I/O to interrupts to vendor-specific instructions. Prototyping and validating all such ancillary behaviors directly in FPGA can easily come to consume the majority of the development effort.

We have observed through profiling [1] that these ancillary behaviors are exceedingly rare during dynamic runtime. This observation leads us to focus our development efforts on common-case behaviors that would actually benefit from the acceleration of FPGA emulation. To that end, we developed an instruction-set and device virtualization mechanism (Figure 1) which combines FPGA emulation and software simulation of different devices and even permits devices (such as the target processors) to dynamically switch between FPGA and software hosts during runtime.



**Figure 1:** In hybrid emulation, FPGA emulation and software simulation together are hosts of the target system. This virtualization of hosting resources is transparent to software in the target system.

As a virtualization mechanism, hybrid emulation allows infrequent behaviors to be omitted from the FPGA hardware while still retaining the abstraction of the complete system. For example, when a “partially implemented” processor on the FPGA encounters an omitted behavior, it can generate a special hardware fault that interrupts instruction execution. This fault is detected and handled by a simulator program on a nearby software host (e.g., PC or SoC hard core). The simulator executes the unimplemented behavior and returns the updated state to the waiting processor on the FPGA. Such a hybrid target proc-

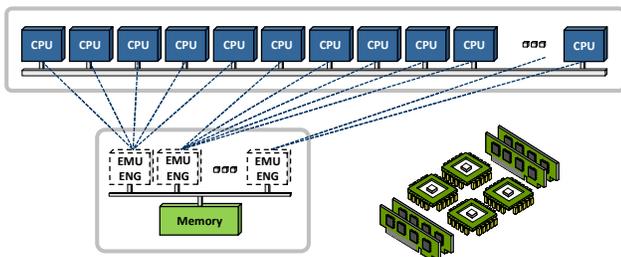
<sup>1</sup> We use “emulation” to refer to modeling the execution of a target computer using FPGAs, as opposed to simulation in software.

<sup>2</sup> <http://www.ece.cmu.edu/~simflex/prototflex.html>

essor emulation can rely on simulation to carry out any unimplemented rare behaviors (e.g., TLB flush, memory-mapped I/O); furthermore, I/O devices can be supported entirely in software simulation. Note that this virtualization of hybrid hosting resources is completely transparent to the target system’s application and operating system. Please refer to [1] for additional discussions on techniques to minimize the implementation effort and emulation performance overhead.

### 3 TIME-MULTIPLEXED EMULATION

Even with FPGA prototyping, emulating a 1000-way MP system by integrating a thousand processors requires significant development and hardware resources. In the case of providing a sufficiently fast functional model, the physical emulation platform need not be as complex as the target system since the underlying host merely has to present the outward behavior and not implement the true structure of the target machine.



**Figure 2:** Decoupling the number of target processors from the number of physical engines.

Time-multiplexed emulation, depicted in Figure 2, virtualizes the host processor emulation resources by mapping multiple processors in the target model to a smaller number of physical “emulation engines”. Each engine can be thought of as a resource implemented in the FPGA that carries out the execution of multiple target processors. Naturally, more engines means faster overall performance. Nonetheless, having fewer engines (at less performance) does not limit the number of target processors that can be modeled. Instead, the integration effort and resources needed when combining multiple engines is determined by the desired aggregate performance and does not reflect the target system’s size.

How many engines would be needed? Contemporary software full-system simulators with instrumentation typically run at 1-10 aggregate MIPS, which limits their use to simulating MP system with only tens of processors. Our goal is to combine as many engines as needed to achieve an aggregate performance practical for hundreds to even thousands of processors. Assuming that a single engine can run at 100 MIPS, ten such engines can be combined in a 1 GIPS emulator, which can functionally model a 100-way system at 10 MIPS per processor, staying within the 100x slowdown limit.

Many approaches are possible for building a time-multiplexed emulation engine. The multithreaded design we adopt keeps each engine as simple as possible. Because each engine hosts multiple processor contexts, we leverage the parallelism between processor instances with an **interleaved** in-order pipeline without data-forwarding or stalls. On each cycle, an instruction from a different processor is issued into the pipeline. This approach simplifies the hardware and is tolerant of long-

latency events such as cache misses, allowing each engine to approach the efficiency of 100MIPS@100MHz.

### 4 APPLICATION OF FUNCTIONAL EMULATION

First, a sufficiently fast (less than 100x slowdown) functional emulator can support non-trivial software development activities for large-scale systems. Even when existing commercial binaries are used (e.g., databases, web servers), fast functional emulation greatly speeds up the many rounds of tuning calibration required to tune these large applications to the hardware configuration. These software activities are excruciatingly slow at simulator speeds but can be accelerated with emulation. Second, a functional emulator can address a key bottleneck in simulation sampling performance studies [2]. The turnaround time in sampling-based studies is dominated by the initial time spent in functional execution to generate checkpointed sampling points and in functional warming of micro-architectural structures (e.g., caches) prior to each measurement. A functional emulator instrumented with cache simulation greatly reduces this cost.

### 5 STATUS AND FUTURE WORK

We have completed a proof-of-concept of hybrid emulation that models a uniprocessor UltraSPARC III workstation running unmodified Solaris 8. The target systems runs on a hybrid host comprising of a Xilinx XUP V2P demo board connected over optimized Ethernet to a Linux PC workstation running Virtutech Simics (<http://www.virtutech.com/>). The frequently exercise behaviors of the UltraSPARC III processor are implemented in FPGA; the remaining processor behaviors are support by embedded software simulation. Simics provides validated software models of I/O devices and the workstation system environment needed for full-system emulation. We have successfully emulated the workstation running uniprocessor workloads with Solaris 8 at speeds of up to 16 MIPS (our first hardware implementation is a multi-cycle processor with  $CPI_{ideal} = 5$ ). Our emulator loads unmodified checkpoints of machine state generated by Virtutech Simics allowing experimentation with existing workload libraries.

We are currently building an interleaved emulation engine supporting up to 16 target processor instances with support for shared-memory. The engine will run multithreaded full-system benchmarks such as SPLASH and TPCC on DB2 and Oracle at 100 MIPS aggregated for the 16 target processor instances. We plan to scale our implementation to tens of engines for simulating 100- to 1000-way systems.

An open question in our work is how to virtualize the physical memory capacity for thousands of processors. We are currently investigating techniques for providing the illusion of large-scale memories by extending available physical memory on FPGAs with slower backing storage (e.g., disk).

### 6 REFERENCES

- [1] E. S. Chung, E. Nurvitadhi, J. C. Hoe, B. Falsafi, K. Mai. ProtoFlex: FPGA-accelerated Hybrid Functional Simulation. CALCM Technical Report 2007-2, Carnegie Mellon University, February 2007. <http://www.ece.cmu.edu/~simflex/publ/2007-02.pdf>.
- [2] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: statistical sampling of computer system simulation. IEEE Micro, 26(4):18-31, Jul-Aug 2006.