RAMP Blue: Implementation of a Manycore 1008 Processor FPGA System

D. Burke, J. Wawrzynek, K. Asanović, A. Krasnov, A. Schultz, G. Gibeling, P.-Y. Droz
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA, USA
email:{drburke,johnw,krste,akrasnov,alschult,gdgib}@eecs.berkeley.edu, droz@ssl.berkeley.edu

Abstract

The RAMP project was undertaken based upon the observation that future computer architectures were likely to rely upon massive parallelism for performance gains, whereas past gains largely leveraged Moore's Law silicon improvements. A need was perceived for an emulation platform to accelerate architecture research and multicore/manycore software engineering, specifically compiler and parallel programming endeavors. The RAMP Blue effort is a conventional direct RTL implementation of a message-passing machine. The system consists of 768– 1008 MicroBlaze cores in 64–84 Virtex-II Pro 70 FPGAs on 16–21 BEE2 boards, surpassing the milestone of 1000 cores in a standard 42U rack. An architecture based on point-to-point channels and switches using a combination of custom and generic hardware provides the basic functionality; virtual-cut-through dimensional routing on one of two hybrid topologies with virtual channels provides the connectivity. A control network with a tree topology provides management and debugging capabilities. A software infrastructure consisting of GCC, uClinux and UPC enables running off-the-shelf applications and scientific benchmarks. Earlier papers documented core and infrastructure elements; this publication describes the system-building effort and evolution of the emulation vision based upon subsequent insights and experience gleaned from the larger systems.

1. RAMP Vision

In 2005, an inflection point occurred in computer development: future products from all major microprocessor manufacturers were henceforth to be single-chip multiprocessors, and anticipated performance improvements would be derived almost exclusively from software-specified parallelism. This development represented a fundament shift in an industry which had benefited from three decades of Moore's Law.

A group of like-minded researchers, in discussing this development at the ISCA conference of 2005, perceived three immediate issues with the traditional development practice:

1. Prototyping any new architecture takes approximately four years and many millions of dollars.
2. Software engineers are ineffective until functional hardware becomes available since simulators are too slow to support development activities.
3. Feedback from software engineers based upon current production hardware arrives too late in overlapping hardware development cycles to affect the succeeding generation, and instead will be reflected in future generations.

In order to address these concerns, a platform which would allow far more rapid evolution than traditional approaches was required. FPGA technology had achieved the capacity and speed to enable a 1024-processor system in only a few dozen FPGAs at very modest cost per processor. Leveraging this fact, RAMP researchers sought to establish a community around a common hardware and software infrastructure, in turn allowing a pooling of resources to build a far more productive environment than could be achieved by disparate efforts. By operating a full RTL model at high-speed, researchers could have much higher confidence in the accuracy of their results and the feasibility of their approach.

Several prototype systems were subsequently built, and as anticipated, are exhibiting sufficient performance to interest the wider research community, including compiler researchers, operating system developers, and distributed system designers. Previous experience has shown that software researchers are only inspired to work with new computer architectures when such hardware prototype is available. Additionally, by reflecting rapid hardware evolution in direct response to software developers feedback, RAMP, unlike conventional chip fabrication, can “tape out” every day and exhibit a near-continual state of flux.

RAMP is a research platform, and can therefore offer capabilities not present in any commercial multiprocessor. As an example, RAMP can demonstrate exactly reproducible behavior, wherein every processor’s memory references and interrupts would happen at exactly the same clock cycle on every run. RAMP is thus finding value for many forms of software experimentation, as well as aiding in software debugging.

A key philosophy of RAMP is to provide a common hardware solution, as opposed to having many institutions build and maintain their own FPGA boards and accompanying software. RAMP stated as a goal that any site should be able to obtain a copy of the hardware at modest cost, and then download a fully-functioning large-scale multiprocessor including hardware design and software environment; currently each PI on the project is equipped with RAMP-1 board hardware, and individual project designs are beginning to be shared freely between various efforts, which in turn acts to lower the barrier to entry for access to a multiprocessor research capability. Beginning with the RAMP-2 (BEE3) platform, dramatic expansion is anticipated in the set of individuals and departments who can participate in this new wave of architecture and software research. Both parallel software and multiprocessor architecture are critical research areas, and RAMP is providing the means to foster and support a much richer interaction between these communities.

RAMP represents a heretofore unavailable combination of speed, accuracy, repeatability, and adaptability in a large-scale multiprocessor development platform. Several sub-efforts are underway, designated by colors: White, Red, Gold, and Purple. Within the overall RAMP project, Blue is oriented toward the most traditional direct RTL implementation. Blue has a stated target of a scalable, multi-board FPGA-based system that would allow construction of up to a 1024-CPU multiprocessor—this size selected as an interesting target because many problems that are invisible at 32 processors and awkward at 128 become glaring at 1024. Furthermore, this scale challenge holds across the entire architecture, network, operating system, and applications disciplines. This milestone was reached and is being disseminated to the software community for further studies specific to the above-mentioned disciplines. This shared artifact consists of hardware and a collection of RTL (”gateware”) available in both conventional HDL as well as Ramp Design language (RDL).

2. Emulation Technologies

Early computer architecture research relied upon convincing argument or basic analytical models to justify design decisions. Beginning in the early 1980’s computers became fast enough that simple simulations of architectural ideas were possible. Since the 1990’s, computer architecture research has come to rely extensively on software simulation. Numerous sophisticated software simulation frameworks exist, including SimpleScalar [1], SimOS [2], RSIM [3], Simics [4], ASIM [5] and M5 [6]. As the field’s research focus shifts to multi-core, multi-threading systems, a new generation of multiprocessor full-system simulators—with accurate OS and I/O support—have recently emerged (e.g., [7], [8], [9]). Software simulation has significantly changed the computer architecture research field; it is relatively easy to use and can be parallelized effectively using separate program instances to simultaneously explore the design space of architectural choices.

Nevertheless, even when evaluating uniprocessor architectures, software simulation is slow to generate a single datapoint. Detailed simulations of out-of-order microprocessors typically execute at kilo-instructions per second. In the case of the multiprocessor simulation, the performance bottleneck is magnified since the simulators slow down as the number of cores rises. A number of researchers have explored mechanisms to speed up simulations. One approach relies upon modifying the inputs used to benchmarks in order to reduce their total running time [10], though the difficulty emerges of finding a set of inputs that accurately mirrors the execution profile of a full run of an application. In another approach, researchers recognized that the repetitive nature of program execution could be exploited to only run a detailed microarchitectural model on a subset of the full program run. The first technique to exploit this was basic block vectors [11], which determines a small set of the whole program code path that provides representative behavior. Later researchers proposed techniques that repeatedly sample program execution to find demonstrably accurate subsets [12]. The drawback of both these approaches to subsetting a full run is that the full application must be run at least once to prepare the samples and associated state checkpoints. Gathering these sample points is slow, and this approach is only feasible if the application benchmarks and inputs are to remain unchanged over the course of many architecture experiments, to allow reuse of the canned execution samples.

As previously discussed, it is now widely accepted that the challenges facing the field will find solutions only by innovating in both hardware and software. The addition of new architectural features requires the development of new application software, languages, compilers, libraries, and operating systems. Even for a fixed architecture, application code should be retuned for any given set of architectural parameters (cache size, memory latency, etc) to enable a fair comparison. Interactive development of software for multiprocessor research is intractable using software-based architectural simulators. In order to engage software researchers, proposed new architecture designs must be fast enough to be usable for real software development.

The possibility of FPGA prototyping and simulation acceleration has piqued the interest of computer architects for as long as the technology existed. FPGAs can be reconfigured to model the components in a target processor design and the resulting logic can be clocked at tens of MHz. Unfortunately, until recently, this avenue has met only limited success due to the restrictive capacity of earlier generation FPGAs and the relative ease of simulating uniprocessor systems in software. An early example of a large-scale FPGA prototyping effort was RPM [13]. The RPM system enabled flexible evaluation of the memory subsystem, but was limited in scalability (8 processors) and did not execute OS code. The renewed interest in FPGA emulation is due to the growth of FPGA capacity to the point where a complex CPU including double-precision floating-point units and caches can comfortably fit on a single FPGA, vastly simplifying model development and raising emulation speed. For the simpler processors anticipated in next-generation "manycore" architectures, several cores can be mapped to a single FPGA and emulation speeds of around 100MHz have been demonstrated [14]. Current FPGA capacity provides the capability for a much needed, scalable research vehicle for full-system multiprocessor research.

Cost rules out a large SMP for most researchers. Also, although an SMP can be used to run multiprocessor applications natively, this does not allow experimentation with different architectural configurations or the addition of new features. Furthermore, when executing natively on real hardware it is difficult to observe, reproduce, and measure low-level system interactions. A large cluster is cheaper way of obtaining multiple hardware cores and can be used to model a multiprocessor system, but is cumbersome and costly to manage and provides only limited observability, reproducibility, and configurability.

The most practical alternative to date has been software simulation, and indeed that has been the vehicle of choice for most architecture research in the last decade. For multiprocessor target systems, simulation speed in total instructions per second diminishes as more target cores are simulated and as operating system effects are included. Further, when detailed timing and power models are incorporated, software simulation slows dramatically. Although techniques to reduce input sizes and sample an application run can reduce simulation time, they are only usable when the software is not changing, and the credibility of results based on simulations of 1000 processor systems running small snippets of reduced-size applications is unclear. Parallelizing simulation runs (e.g., when performing a parameter sweep) can provide increased simulation throughput, but the amount of memory, and hence cost, of each node in a host compute cluster rises rapidly (each host compute node must have sufficient memory to model an entire target system). As mentioned above, software developers rarely use software simulators since they run too slowly, and neither sampling nor parallelizing across independent runs helps with software development, which is primarily constrained by the latency to complete a single run with a new version of the application code.

An unexpected result is that is has proven difficult to parallelize the software simulation of a parallel target system to take advantage of a multiprocessor host system, even if the software simulator is well structured [15]. For detailed cycle-level models, each component must communicate data with each other component on every clock cycle, and this degree of synchronization and communication results in poor performance on current multiprocessors, which were designed to run single-threaded applications well (perhaps future multicore systems will have better synchronization and communication primitives, and hence improve their ability to run simulations of proposed new multicore designs).

A few groups have completed custom chip prototypes of their proposed architectural ideas. This is an expensive option that experience has shown takes around five years to complete a working prototype of a given set of architectural mechanisms. Although this provides a highly credible proof of concept, there is usually limited ability to experiment with the choices made (although most research prototypes typically include more support for experimentation than a commercial design). Also, the design budget limitations usually result in a prototype that is considerably slower than a contemporary commercial product.

Architecture-level FPGA emulation provides a compromise between these alternatives. It is so much cheaper than custom hardware that small groups can afford highly scalable systems; it is as flexible as simulators so that the state of the art in parallel computing can be rapidly evolved; and it is so much faster than simulators that software researchers can be tempted to explore new hardware ideas. The credibility of FPGA emulation can be much higher than for software simulation, both because entire applications can be run under complete software stacks and also because the model construction is more similar to hardware design and can actually reuse hardware designs as part of a model. Surprisingly, modern FPGA designs have clock rates only around 2–4 times slower than contemporary custom chip research prototypes. This is mainly due to the fact that off-the-shelf FPGAs use process technology that is two or three generations ahead of that available for research prototypes and FPGA designs can be completed more rapidly and moved more easily to new technologies.

A related, but somewhat different approach is the use of FPGAs and special hardware to accelerate gate-level emulation for chip verification. This provides a useful tool for chip verification, but operates at too low a level to be useful for early stage architecture exploration. In order to span the range of possibilities, RAMP Blue was developed which more closely resembles this approach in concept. In contrast, the style of FPGA emulation being proposed for RAMP White, and especially RAMP Gold, is at a higher-level of design representation, which reduces the FPGA capacity requirements, increases emulation throughput, and reduces the design effort required to experiment with a new design.

3. Berkeley Emulation Engines

From the outset, there was a common desire to not begin the RAMP project by designing yet another FPGA board, but to rather adopt (at least for initial research needs) the Berkeley Emulation Engine (BEE2 [16]) for the RAMP-1 system. In that capacity, the BEE2 has served well as the platform of the early RAMP machine prototypes and greatly informed the feature list for the next generation board. The follow-on BEE3 hardware platform, which has completed the design and prototyping phase and entering production, is based on a board design employing Virtex-5 FPGA architecture, which is two FPGA generations newer that the BEE2 component Virtex II Pro. The new RAMP-2 hardware (BEE3) is scheduled to enter full-scale production by 2H 2008, and contains numerous features and capabilities specific to the RAMP project.

The genus platform, the BEE1, was primarily designed to serve as an ASIC emulator and test bed for circuit design [17] to fulfill an ongoing need at the Berkeley Wireless Research Center. The platform demonstrated significant usefulness outside of the original domain, however, and it became obvious that inclusion of certain features in future versions would prove fruitful.

The BEE2 is the second generation multiple FPGA board, and intentionally targets an increased range of applications such as real-time signal processing, scientific computing, and high performance reconfigurable computing. This broader domain is reflected in the BEE2's architecture, which is shown in Figure 1. As the RAMP effort began, packaging the BEE2 for high-density computing became important. Hence, the 2U chassis and mechanical components, along with a high-efficiency internal power supply were designed and fabricated dictated by RAMP requirements.

The main board contains five Xilinx Virtex- II Pro FPGAs with footprints capable of handling the largest parts then available (70 and 100 series) for maximum I/O capability. Internally, the Virtex-II Pro 70 contains a substantial number of general purpose reconfigurable logic (33,088 slices with a total of 66,176 4-LUTs and flip-flops), built in multipliers (328 18x18 bit), large SRAM block RAMs (328 dual-ported 18 Kb SRAM), highly configurable I/O pins, high-speed serial transceivers (20 MGT channels), and two hard PowerPC cores with built-in L1 cache and a peak core clock of 400 MHz.

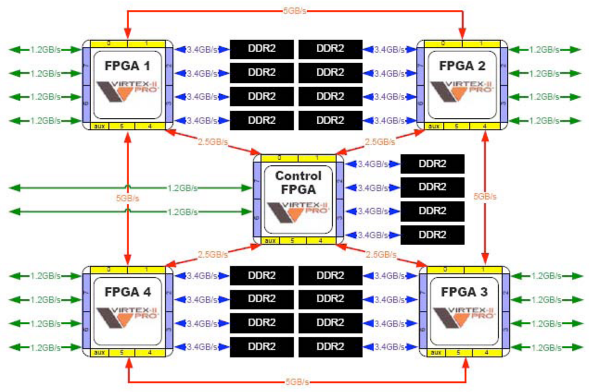


Figure 1

On the board, each of the FPGAs is mapped to four independent 72-bit DDR2 banks, capable of a peak throughput of 3.4 GBps per channel. Four of the

FPGAs are designated as "user" FPGAs, and are connected in a ring topology with a peak throughput of 5 GBps between each adjacent pair. The fifth FPGA, termed the "control" FPGA, is connected in a star topology with each user FPGA with a peak communication capability of 2.5 GBps. Individual user FPGAs also have four sets of bonded multi-gigabit transceiver (MGTs) forming four independent 10 Gbps high-speed serial I/O channels off-board, while the control FPGA has two of the bonded channels. The bonded high-speed serial channels run to CX4 connectors which allow Infiniband, 10 Gb Ethernet, or simple point-to-point XAUI based connections over fiber or copper cables

The RAMP2 platform will be based on the BEE3, the third generation Berkeley FPGA based computer system. The module was designed as a collaboration between UC Berkeley and Microsoft Research and will be manufactured and supported by BEEcube, Inc.

The BEE3 design was specifically optimized for system emulation and simulation acceleration of multi-processor computer systems, FPGA based computational acceleration in High Performance Computing (HPC), high speed digital signal processing (e.g. Radio Astronomy, SETI), ASIC/SoC real-time emulation, and DSP/Communication algorithm real-time prototyping. The BEE3 utilizes devices from the Xilinx 65nm FPGA family: each 2U rack-mount BEE3 module consists of four Virtex-5 LXT/SXT/FXT FPGA chips, along with up to 64GB DDR2 DRAM and eight 10Gigabit Ethernet interfaces for inter-module communication. In addition, up to 4 PCI Express x8 connections allow a maximum of 16GB per second full-duplex data communication between each BEE3 module and a front-end host computer. At a power consumption of less than 400 watts, each BEE3 module can provide over 4 trillion integer operation per second, or emulate over 64 RISC processor cores concurrently at a several hundred MHz rate. Each of the four FPGA provides significant capability improvement over the BEE2 (Figure 2):

- Dual channel DDR2-400/533/667 RDIMMs, two DIMMs per channel, with a maximum of 4GB DRAM capacity per DIMM, for a total of 16GB total per FPGA.
- Dual 10GBase-CX4 Ethernet interfaces.
- A single PCI-Express x8 end-point slot.
- Direct 40 LVDS high-speed connection to external multi-GHz analog converter devices, such as ADC and DAC.
- A number of system management components include Gigabit Ethernet, RTC/EEPROM, RS232 serial port, and SD Card.

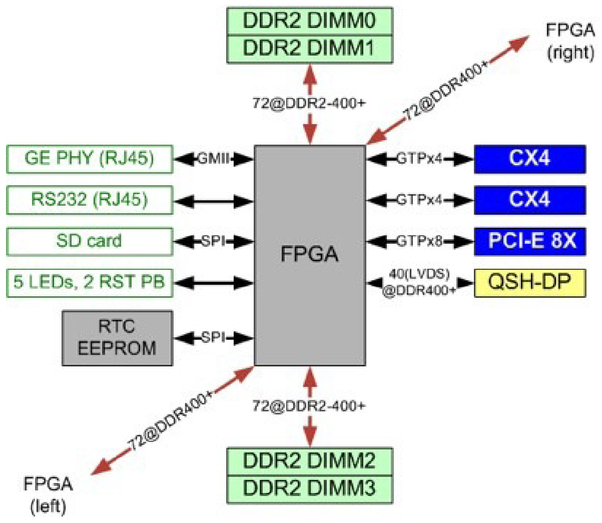


Figure 2

Prior to the production release, single-chip development boards are utilized to accelerate IP design in parallel with BEE3 hardware efforts (ML505 and V5 XUP). Moreover, the lower-cost and lower capacity hardware provides a means for external collaboration and participation by outside entities on a low-risk trial basis. Because both the BEE3 and ML505 use the same Virtex 5 FPGA components, any gateware designs are completely fungible, albeit with some capacity constraints.

4. RAMP Hardware

RAMP Blue, first of five emulation platforms explored, is an direct-RTL/RDL emulated message-passing machine which can be used to run parallel applications written for the Message-Passing Interface (MPI) standard, or for partitioned global address space languages such as Unified Parallel C (UPC). Although MPI is not an ideal candidate for manycore communication, this choice eased implementation and permitted execution of existing codes, which was a primary goal. RAMP Blue can also be used to model a networked server cluster. In tangible realizations of the RAMP hardware, there have been three major system rollouts: 768, 1008, and 256 core (production grade). RAMP Blue is an exemplar of a classical RTL direct-mapped approach and has proven to be a rapidly evolving, highly scalable, and very successful demonstration vehicle. What follows is a brief overview of the physical apparatus.

The first RAMP Blue prototype was developed at UC Berkeley in August 2006, shown in Figure 3. It comprises a collection of RAMP-1 boards housed in preliminary 2U chassis and assembled in a standard 19” rack with external supplies. Physical connection among the eight boards is through 10 Gbps Infiniband cables (light blue cables Figure 3).

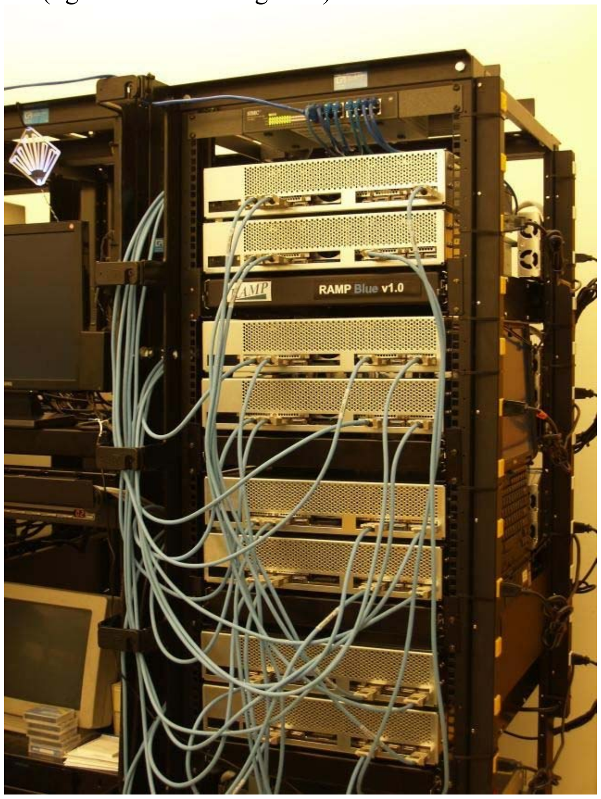


Figure 3

The RAMP-1 boards are wired in an all-to-all configuration with a direct connection from each board to all others through 10 Gbps links. System configuration, debugging, and monitoring are through a 100 Mbps Ethernet switch with connection to the control FPGA of each board (dark blue wires in the figure). For system management and control, each board runs a full-featured Linux kernel on one PowerPC 405 hard core embedded in the control FPGA. Initial target applications include the UPC versions of the NASA Advanced Supercomputing (NAS) Parallel Benchmarks. The four user FPGAs per RAMP-1 board are configured to hold a collection of 100MHz Xilinx MicroBlaze soft processor cores running uCLinux. This initial version mapped eight processor cores per FPGA. The first prototype, with 32 user FPGAs, emulates a 256-way cluster system. The number of processor cores is scaled up through several means. More RAMP-1 boards were added—the simple all-to-all wiring configuration will accommodate up to 17 boards. More cores are desirable per FPGA—this configuration of eight processor cores per FPGA only consumed 40% of the FPGA’s logic resources. Ideally, the target was up to 16 MicroBlaze cores per FPGA, and 1024 cores in a system. All necessary multiprocessor components are implemented within the user FPGAs. In addition to the soft processor cores, each FPGA also contains a packet network switch (one for each core) for connection to units on the same and other FPGAs, shared memory controllers, shared double-precision floating-point units (FPUs), and a shared “console” switch for connection to the control FPGA. In RAMP Blue, each processor is assigned its own DRAM memory space (at least 250MB per processor). The external memory interface of the MicroBlaze L1 cache connects to external DDR2 DRAM through a memory arbiter, as each DRAM channel is shared among a set of MicroBlaze cores. Since each RAMP-1 user FPGA has four independent DRAM memory channels, four processor cores would share one channel in the maximum-sized configuration (16 processor cores per FPGA).

With each processor operating at 100MHz and each memory channel signaling at a 200MHz DDR 72-bit data rate, each processor can transfer 72 bits of data at 100 MHz, which is more than each processor core can consume even in the maximum-sized configuration. A simple round-robin scheme is used to arbitrate among the cores. The processor–processor network switch currently uses a simple interrupt-driven programmed I/O approach. A Linux driver provides an Ethernet interface so applications can access the processor network via traditional socket interfaces. A next generation network interface is planned with direct memory access through dedicated ports into the memory controller. A 256-core (8 per FPGA) version of the RAMP blue prototype has been fully operational running the NAS Parallel Benchmark suite (all class S), since December 2006.

4.1. MicroBlaze

The processor selection process identified the Xilinx MicroBlaze as the optimal soft core for this project primarily based on resource utilization. The MicroBlaze is a RISC processor optimized for implementation in Xilinx FPGAs and having a basic 32-bit ISA, a three-stage pipeline, a 32-word register file, and a Harvard-style direct-mapped L1 cache with configurable size. The core can be customized to include a single-precision FPU that shares the existing register file, a hardware multiplier, divider and barrel shifter, several CISC instructions, and several exceptions. The MicroBlaze also has considerable software support available, including a GCC backend and a port of uClinux [18]. Access to this infrastructure was instrumental in meeting the goal of running off-the-shelf code and benchmarks. The MicroBlaze supports several interfaces including the IBM CoreConnect On-chip Peripheral Bus (OPB), Xilinx Local Memory Bus (LMB), Xilinx CacheLink (XCL), and Xilinx Fast Simplex Link (FSL). The FSL is a simple FIFO-like interface that provides unidirectional point-to-point connections, which was particularly appropriate for this effort.

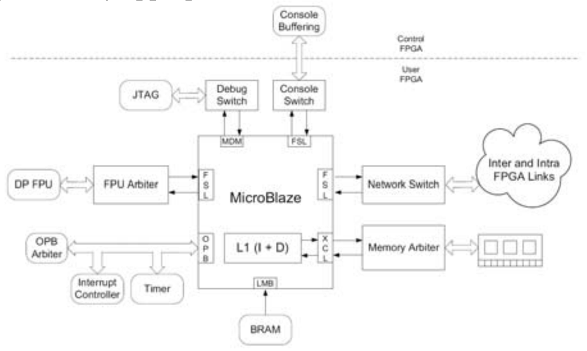


Figure 4

The FSL is used for the majority of the peripherals, though the LMB and the OPB are required for the dedicated block RAM, and the interrupt controller and timer, respectively (Figure 4). The use of the FSL was motivated by its point-to-point nature, which was a good philosophical and syntactic match to the RAMP Description Language (RDL) [19] for research and debugging purposes.

4.1.1. Shared Floating-Point Unit

Most off-the-shelf applications and scientific benchmarks targeted by RAMP Blue require double-precision floating-point arithmetic. A double-precision FPU is unavailable for the MicroBlaze core, and the integrated single-precision FPU is spatially prohibitive at 1000 slices per instantiation. Thus, a custom, shared double-precision, FPU was designed and implemented for those tasks.

In order to minimize the complexity of implementing and verifying an IEEE-compliant double-precision FPU, the arithmetic unit utilizes existing Xilinx library LogiCORE floating-point blocks for addition, multiplication, division, and comparison. During operation, the FPU arbitrates among a parametrizable number of FSL-based MicroBlaze interfaces, one for each client. Operations are requested as four 32-bit FSL writes, with the control bits encoding the operation. A de-serializer assembles the two 64-bit operands and requests the correct functional unit. When access to the input bus is granted, the operands are transferred into the functional unit. Upon completion, and after output bus access is granted, the result is transferred to the serializer. Two 32-bit FSL reads by the MicroBlaze then return the 64-bit result. Exceptions are returned as silent NaN values. Figure 5 shows an overview of this architecture.

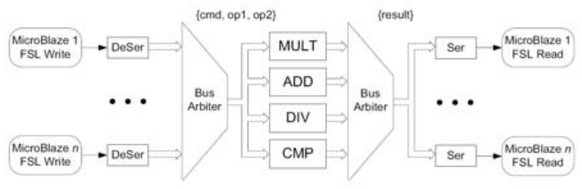


Figure 5

In addition to good fabric utilization, the shared FPU also improves computational efficiency. In this case of a heavily pipelined FPU and a lightly pipelined blocking MicroBlaze, providing each core with a dedicated FPU would result in idle compute cycles. This implementation allows multiple MicroBlaze cores to leverage FPU pipelining by using multi-threading, achieving high overall utilization of the shared resource.

4.1.1. uClinux Operating System

Each MicroBlaze core runs an instance of uClinux to achieve the goal of running GCC-compiled, off-the-shelf applications. uClinux is a variant of the Linux kernel designed for microcontrollers without memory-management units. As a consequence, uClinux provides neither virtual memory nor memory protection. However, most of the kernel support for device drivers, networking and file systems is present under uClinux. The POSIX API is fully supported with the exception of the fork() and brk() system calls, and an extensive user application and library distribution is also available. Both the kernel and the user applications build successfully using the GCC MicroBlaze backend provided by the open source community. In order to meet the goal of running standard scientific benchmarks, the NAS Parallel Benchmarks (NPB) suite [20] were selected and implemented in the Unified Parallel C (UPC) framework [21]. UPC was elected over the more common MPI because it functions at a higher level, and because this particular UPC implementation was developed at UC Berkeley, making the authors available for ready consultation.

The most challenging part of porting the UPC framework to a new platform is ensuring that its communication layer, Global-Address-Space Networking (GASNet) [22], works properly with both the kernel and the underlying communication hardware. GASNet provides message-passing conduits implemented over specific networking hardware or generic UDP sockets, with RAMP Blue using the latter mechanism for simplicity. Future RAMP systems should gain a significant amount of performance by implementing custom Remote DMA (RDMA) network hardware and a corresponding GASNet conduit. Such an implementation will decouple computation and communication and eliminate the overhead associated with the UDP/IP stack and uClinux networking.

5. Network

5.1.1. Crossbar

In RAMP Blue, messages are passed along a custom network composed of intra-FPGA, intra-BEE2 and inter-BEE2 links. High-level design choices were made to simplify the network design and implementation:

- **Routing**: Packets are statically (non-adaptively) source routed at each hop in the network, with broadcast not supported.
- **Topology**: Chip-level crossbar embedded in a board-level mesh embedded in a system-level all-to-all graph, or chip-level crossbar embedded in a system-level 3D mesh.
- **Packet Format**: The gateware is oblivious to packet format except for the route header, added by the source and stripped by the packet buffers. For compatibility, the packet format is Ethernet II encapsulated in a small amount of control information.
- **Delivery Guarantees**: Delivery is guaranteed end-to-end in software. The gateware does not perform check summing or re-transmission.
- **Flow Control**: Virtual cut-through, blocking only if the next buffer is unavailable.

Network Gateware

The switch is comprised of two basic elements: a buffer unit and a crossbar switch. Buffer units store a single packet at each hop in the network and make requests to the switch for the next hop. Each buffer unit uses a single BRAM to provide buffering for an MTU of up to 2048 bytes and exposes FSL read and write interfaces with additional signals used for arbitration. Packets are tagged with start and end control bits, allowing for error tolerance. The crossbar switch unit is fully parameterized in the number of ports, data width, and latency. Adjusting the data width trades performance for resource utilization, balancing between overall performance optimization and a reduced design place-and-route cycle. For each output buffer unit, the switch arbitrates among the inputs servicing that output using a starvation-free round-robin policy. Upon winning arbitration, the entire packet is transmitted.

Reliability and Deadlock

To simplify the implementation, the amount of error checking in the network is sufficient only to guarantee the proper operation of the network itself. Bit errors in the route header 58 can result in misrouted packets. We rely on de-serialization and next-hop checking to discard such packets in the network and destination checking to discard such packets at the endpoint. Corrupted start or end bits can also result in discarded packets. The low bit error rates observed in the lab suggest that these should be relatively rare occurrences.

Multi-processor networks admit the possibility of deadlock, which occurs when all buffers in a cycle become full. With virtual cut-through, a common solution is to partition the buffers and place a partial ordering on them [23]. Another popular method is to use virtual channels [24]. In RAMP Blue, the all-to-all topology requires a combination of these methods to avoid deadlock in all cases. A partial ordering on the buffers is enforced using dimensional routing, with sufficient dimensions provided by virtual channels. The cost of this implementation is two additional receive buffers on the intra-board links out of a total of 38 buffers and a corresponding increase in the crossbar-switch ports.

MicroBlaze Interface

Each MicroBlaze core exposes a network-visible FSL interface. Packets are transmitted by first testing buffer unit availability with a tentative non-blocking FSL write, and then upon success, copying the balance of the packet without blocking. When a packet is available for receiving, the buffer generates an interrupt, instructing the MicroBlaze to copy the packet using a series of FSL reads. This interface is simple to implement, but suffers from low-performance. A future design using a small DMA engine is expected to greatly improve the performance of the entire RAMP Blue system. The network driver is almost exactly the same as the driver used for the control-network link. The driver provides an Ethernet-device abstraction to the operating system, pre-pending the source route before dispatch. The source routes are computed in a distributed fashion and conveyed to the driver using custom ioctl system calls by a parameterized network-setup program running on each individual node at boot time. The routes are thus static and not adaptive to link failure, such as an unanticipated cable disconnection.

5.1.2 Network Topology

Board-to-board connections are implemented using copper 10GBASE-CX4 cables, which provide full-duplex 10 Gbps links between FPGAs. Although having a theoretic high bandwidth, the latency of these links (tens to hundreds of cycles) is large relative to that of intra-board LVCMOS links (two to three cycles).

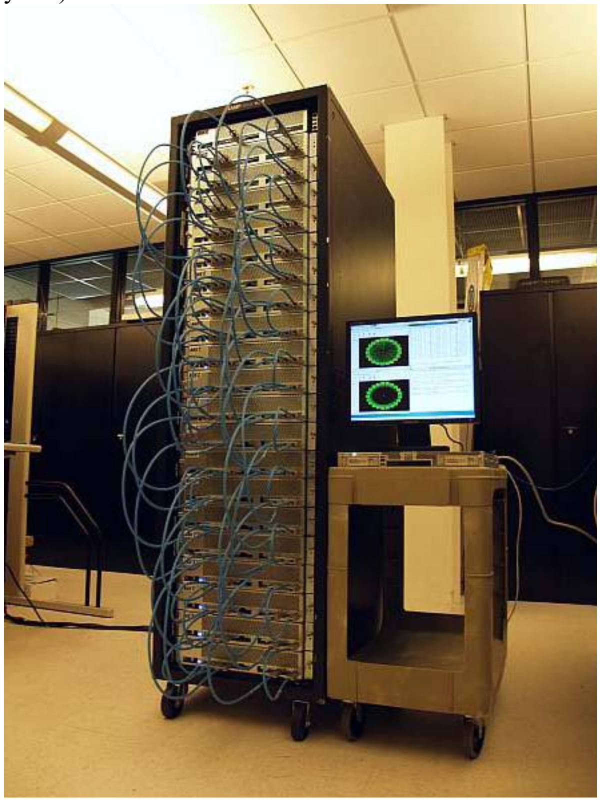


Figure 6

For up to 16 BEE2 modules, an all-to-all topology was favored, with each module having one high-speed serial connection to every other module. The primary advantage of this topology is that it minimizes the number of inter-board links along any communication path, thereby optimizing latency and reliability. The path from any one soft core to another requires at most four intra-board (two on each board) and one inter-board links. The FPGA and port assignments are rotated between chassis so as to use only vertically routed cables, however due to port limitations, this scheme does not scale beyond 17 modules.

For over 18, and up to the maximum 21 modules (Figure 6) which can be physically accommodated, a 3D-mesh topology is employed; the FPGA and port assignments are again rotated in order to match the constraints of the rack and incorporate only vertically routed cabling.

6. Results

With respect to the highest-density, single-FPGA implementation, a full 12-core RAMP Blue design consumes 32,991 slices (99%), 61,891 LUTs (93%), 37,198 flip-flops (56%), and 181 block RAMs (55%). These results assume the following processor options: 90 MHz core clock, no optional functional units, all optional exceptions, 2 KB I-Cache, 8 KB D-Cache, LMB block RAM, and OPB peripherals.

Because the BEE2 system board clock is 100 MHz, it was expedient to inject the 90 MHz clocking signal from external hardware, which could prove cumbersome for external collaborators. An optimized 8-core (numbered circles in figure) version was developed for the production release, and the relaxed constraints enabled use of the on-board clocking tree (Figure 7).

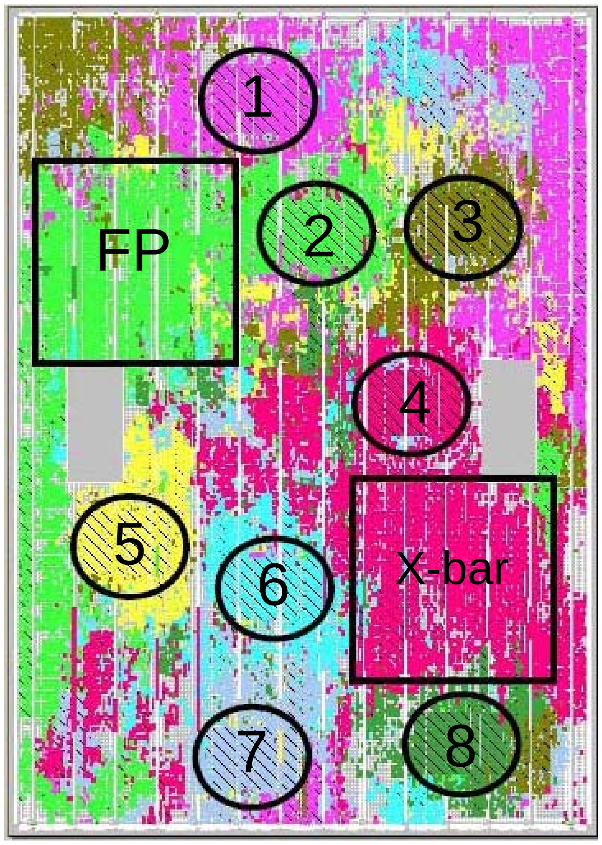


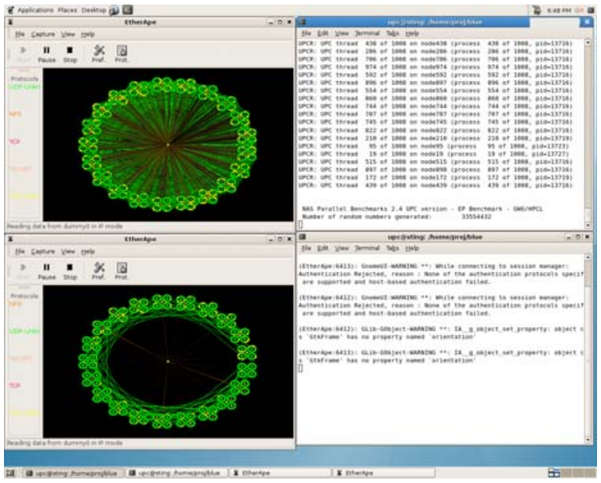
Figure 7

The infrastructure consists of three DDR2 controllers, four XAUI blocks, double-precision FPU, and 8-bit network buffers and crossbar switch. Based on these results, it is anticipated that a configuration with 16 MicroBlaze cores and four DDR2 controllers is feasible with extensive optimizations.

The synthetic Whetstone benchmark indicated an FPU performance of 2.5 to 3 MFLOPS, corresponding to an overall system performance of 2–3 GFLOPS for 768–1008 MicroBlaze cores. Although Whetstone is not a realistic benchmark, it is useful in noting the difference in performance between the software-emulated and the hardware-accelerated FPUs. In this test, it showed a speedup of 15 for scores of 200 KFLOPS and 3 MFLOPS, respectively. We used the Netperf benchmark [25] to determine the performance of the MicroBlaze-to-MicroBlaze network for user code running under uClinux. Netperf tests bulk-transfer throughput and round-trip response time using both TCP and UDP. There is no significant variation in either throughput or response time for the different link combinations for single communicating pairs. In contrast, as the payload size grows, there is a clear increase in round-trip response time. This result suggest that the software overhead of transmitting and receiving is dominating the effects of latency in different links, even with the inter-board links having a much greater latency than the intra-board links do. Thus, there is an urgent need for a DMA-based network interface and driver.

The goal of RAMP Blue was to create a message-passing multi-core system, capable of running off-the-shelf applications and scientific benchmarks, on the BEE2 platform. The current implementation meets these goals. RAMP Blue is currently able to run uClinux on 768–1008 independent MicroBlaze cores on 16–21 BEE2 boards. In operation in Figure 8, each green dot represents a single core, and lines between processing elements represent traffic sampled at nominally 5%. The cores are able to run the majority of the NAS Parallel Benchmarks and compute accurate double-precision floating-point values.

Beyond the obvious functional correctness, RAMP Blue represents a truly scalable, portable, and capable research system fully accessible to software developers. By being available today, compiler and application work for thousand-core devices can being without delay. RAMP Blue will further server as the first step in more advanced emulation technologies under study.



```
UPCR: UPC thread  438 of 1008 on node438 (process  438 of 1008, pid=13716)
UPCR: UPC thread  286 of 1008 on node286 (process  286 of 1008, pid=13716)
UPCR: UPC thread  786 of 1008 on node786 (process  786 of 1008, pid=13716)
UPCR: UPC thread  974 of 1008 on node974 (process  974 of 1008, pid=13716)
UPCR: UPC thread  592 of 1008 on node592 (process  592 of 1008, pid=13716)
UPCR: UPC thread  896 of 1008 on node897 (process  896 of 1008, pid=13716)
UPCR: UPC thread  554 of 1008 on node554 (process  554 of 1008, pid=13716)
UPCR: UPC thread  860 of 1008 on node860 (process  860 of 1008, pid=13716)
UPCR: UPC thread  744 of 1008 on node744 (process  744 of 1008, pid=13716)
UPCR: UPC thread  787 of 1008 on node787 (process  787 of 1008, pid=13716)
UPCR: UPC thread  745 of 1008 on node745 (process  745 of 1008, pid=13716)
UPCR: UPC thread  822 of 1008 on node822 (process  822 of 1008, pid=13716)
UPCR: UPC thread  210 of 1008 on node210 (process  210 of 1008, pid=13719)
UPCR: UPC thread   95 of 1008 on node95 (process   95 of 1008, pid=13723)
UPCR: UPC thread   19 of 1008 on node19 (process   19 of 1008, pid=13727)
UPCR: UPC thread  515 of 1008 on node515 (process  515 of 1008, pid=13716)
UPCR: UPC thread  897 of 1008 on node898 (process  897 of 1008, pid=13716)
UPCR: UPC thread  172 of 1008 on node172 (process  172 of 1008, pid=13719)
UPCR: UPC thread  439 of 1008 on node439 (process  439 of 1008, pid=13716)


 NAS Parallel Benchmarks 2.4 UPC version - EP Benchmark - GWU/HPCL
 Number of random numbers generated:          33554432
```

```
(EtherApe:6413): GnomeUI-WARNING **: While connecting to session manager:
Authentication Rejected, reason : None of the authentication protocols specif
 are supported and host-based authentication failed.

(EtherApe:6412): GnomeUI-WARNING **: While connecting to session manager:
Authentication Rejected, reason : None of the authentication protocols specif
 are supported and host-based authentication failed.

(EtherApe:6412): GLib-GObject-WARNING **: IA__g_object_set_property: object c
s `GtkFrame' has no property named `orientation'

(EtherApe:6413): GLib-GObject-WARNING **: IA__g_object_set_property: object c
s `GtkFrame' has no property named `orientation'
```

Figure 8

7. Acknowledgements

The RAMP collaboration has been funded in part by the National Science Foundation, grant number CNS-0551739. Special thanks to Xilinx for their continuing financial support and donation of FPGAs, and development tools. We appreciate the financial support provided by the Gigascale Systems Research Center (GSRC). Thanks to IBM for their financial support through faculty fellowships and donation of processor cores, and to Sun Microsystems for processor cores. This project has been a collaborative effort between universities and companies. The other senior investigators are David Patterson (UC Berkeley), Mark Oskin (U Washington), Shih-Lien Lu (Intel), Christoforos Kozyrakis (Stanford), James C. Hoe (CMU), Derek Chiou (UT Austin), and Joel Emer (Intel and MIT).

There is an extensive list of industry and academic friends who have given valuable feedback and guidance. Here we especially give thanks to Arvind (MIT) and Jan Rabaey (UCB) for their advice. The work presented in this paper is the effort of the RAMP students and staff: Hari Angepat, Jared Casper, Chen Chang, Martha Mercaldi, Nju Njoroge, Andrew Putnam, and Sewook Wee. Special thanks to Chuck Thacker (Microsoft Research) for his contributions and hard work on the design of the BEE3 hardware platform.

8. References

[1] D. Burger and T. M. Austin. "*The Simplescalar Tool Set, Version 2.0*", Technical Report 1342, Computer Sciences Department, University of Wisconsin, Madison, June 1997.

[2] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. "Complete computer system simulation: The SimOS Approach". *IEEE Parallel and Distributed Technology: Systems and Applications*, Winter 1995, 3(4):pp 34–43.

[3] V. S. Pai, P. Ranganathan, and S. V. Adve. "*RSIM Reference Manual, Version 1.0*". Technical Report 9705, Electrical and Computer Engineering Department, Rice University, July 1997.

[4] Virtutech, Simics.

[5] J. Emer, P. Ahuja, E. Borch, A. Klauser, C.-K. Luk, S. Manne, S. S. Mukherjee, H. Patil, S. Wallace, N. Binkert, R. Espasa, and T. Juan, "ASIM: A Performance Model Framework". *IEEE Micro*, Feb 2002.

[6] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems". *IEEE Micro*, 2006, 26(4):pp 52–60.

[7] Simflex: Fast, Accurate & Flexible Computer Architecture Simulation",
www.ece.cmu.edu/simflex/flexus.html.

[8] M.M.K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution Driven Multiprocessor Simulator (GEMS) Toolset", *Computer Architecture News*, 2005.

[9] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt, "Network-oriented Full-system Simulation Using M5", *Proceedings of Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads*, February 2003.

[10] A. KleinOsowski and D. Lilja, "*Minnespec: A New SPEC Benchmark Workload for Simulation-based Computer Architecture Research*", 2002.

[11] T. Sherwood, E. Perelman, and B. Calder. "*Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications*", 2001.

[12] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, "*Smarts: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling*", 2003.

[13] K. Oner, L. A. Barroso, S. Iman, J. Jeong, K. Ramamurthy, and M. Dubois, "The Design of RPM: An FPGA-based Multiprocessor Emulator", *Proc. 3rd ACM International Symposium on Field-Programmable Gate Arrays (FPGA'95)*, February 1995.

[14] A. Krasnov, A. Schultz, J. Wawrzynek, G. Gibeling, and P.-Y. Droz. "RAMP Blue: A Message-passing Manycore System in FPGAs". *Field Programmable Logic and Applications, 2007 (FPL 2007). International Conference on,* 27-29 Aug. 2007, pp 54-61.

[15] K. C. Barr, R. Matas-Navarro, C. Weaver, T. Juan, and Joel Emer, "Simulating a Chip Multiprocessor with a Symmetric Multiprocessor. *Boston Area Architecture Workshop*, Providence, RI, January 2005.

[16] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: A High-End Reconfigurable Computing System," *IEEE Design and Test of Computers*, vol. 22, no. 2, 2005.

[17] C. Chang. "*Hardware Design and Implementation of BEE: A Real-Time Hardware Emulation Engine*", Master's thesis, University of California, Berkeley, 2002.

[18] J. Williams, "MicroBlaze uClinux Project Home Page,"
http://www.itee.uq.edu.au/˜jwilliams/mblaze-uclinux/.

[19] G. Gibeling, A. Schultz, J. Wawrzynek, and K. Asanovic, "*The RAMP Architecture, Language, and Compiler*," University of California, Berkeley, Technical Report, 2007.

[20] R. van der Wijngaart, "*NAS Parallel Benchmarks Version 2.4*," National Aeronautics and Space Administration, Technical Report NAS-02-007, 2002.

[21] W. Carlson et al., "*Introduction to UPC and Language Specification*", Center for Computing Sciences, Institute for Defense Analyses, 1999.

[22] R. A. Jones, "*Netperf: A Network Performance Benchmark (Revision 2.0)*," Hewlett-Packard Company, Technical Report, 1995.

[23] K. Gunther, "Prevention of Deadlocks in Packet-Switched
Data Transport Systems," *Communications, IEEE Transactions on* [legacy, pre-1988], vol. 29, no. 4, pp. 512–524, 1981.

[24] W. Dally and C. Seitz, "Deadlock-free message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. 36, no. 5, 1987, pp. 547–553.

[25] D. Bonachea, "*GASNet Specification, v1.1*," Lawrence Berkeley National Laboratory, Technical Report CSD-02-1207, 2002.