To appear in FPGA'95

The Design of RPM: An FPGA-based Multiprocessor Emulator

Koray Öner, Luiz A. Barroso, Sasan Iman, Jaeheon Jeong,
Krishnan Ramamurthy and Michel Dubois

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
oner@paris.usc.edu

Abstract

Recent advances in Field-Programmable Gate Arrays (FPGA) and programmable interconnects have made it possible to build efficient hardware emulation engines. In addition, improvements in Computer-Aided Design (CAD) tools, mainly in synthesis tools, greatly simplify the design of large circuits. The RPM (**R**apid **P**rototype Engine for **M**ultiprocessors) Project leverages these two technological advances. Its goal is to develop a common hardware platform for the emulation of multiprocessor systems with different architectures.

For cost reasons, the use of FPGAs in RPM is limited to the memory controllers, while the rest of the emulator, including the processors, memories and interconnect, is built with off-the-shelf components. A flexible non-intrusive event logging mechanism is included at all levels of the memory hierarchy, making it possible to monitor the emulation in very fine detail. This paper presents the hardware design of RPM.

Keywords: Field-Programmable Gate Arrays (FPGAs), message-passing multicomputers, shared-memory multiprocessors, rapid prototyping, logic emulation.

1. INTRODUCTION

A hardware design project usually goes through the following steps: project specification, design, simulation, prototyping, and production. Software simulators are inexpensive and very flexible. However, they are very slow when some degree of accuracy is desired. On the other hand, prototypes are very expensive and typically inflexible. As the complexity of a system design increases, the efficiency of its software simulator decreases and the cost of its prototype increases. The gap between these two steps in the design of a complex system presents a major challenge to designers. Moving too early to prototyping may prove costly as prototypes are hard to modify. Extending the simulation phase for too long delays the schedule while not guaranteeing a correct design. An attractive option is to build a prototyping testbed which is flexible enough to emulate different systems so that its cost is amortized as it is re-used. Recent improvements in Field-Programmable Gate Array (FPGA)[13] technology along with the emergence of programmable interconnects (such as APTIX[1] FPICs) have enabled the construction of flexible emulators. Quickturn's Enterprise [14], PiE architectures [8] [11] and INCA's Virtual ASIC II Logic Emulator [7] are good examples of such systems.

RPM is a configurable hardware platform to emulate the different models of asynchronous MIMD (Multiple Instructions Multiple Data streams) multiprocessors with up to eight processing elements. In MIMD systems processors run on different clocks and execute their own instructions. Every processing element consists typically of a CPU, some cache and a share of the system memory. Processing elements are connected by fast networks such as buses, crossbars or meshes. The main difference between RPM and other emulation systems is that RPM is geared towards a specific type of system, i.e., an MIMD multiprocessor, whereas current emulators (such as Quickturn's Enterprise) are mainly geared to ASIC emulation. In RPM the controllers in each level of the memory hierarchy are built with FPGAs and the rest of the system is built with off-the-shelf components.

RPM is faster and more detailed than software simulators. Consequently, it can run more realistic programs and it yields more accurate performance evaluation and design validation. RPM is a real computer with its own I/O and the code running on it is not instrumented (as is the case with simulators). As a result, it "looks" exactly like the target machine to the programmer and it can run general purpose programs, including binaries from production compilers,

This research was supported by the National Science Foundation under Grant MIP-9223812.
Future developments on this project will be posted on WWW at http://www.usc.edu/dept/ceng/dubois/RPM.html.

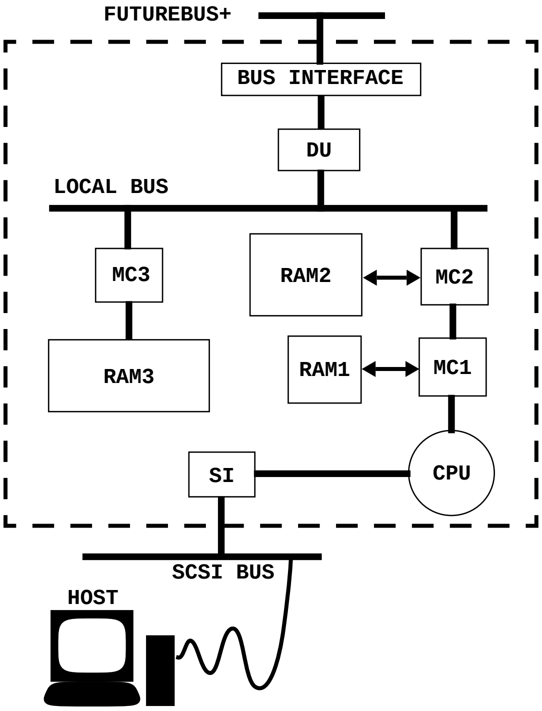
operating systems, and software utilities.

In the following, we first describe the architecture of RPM. In Section 3 we list some of the target architectures that can be emulated with RPM. In Section 4 we explain the design methodology. In Section 5 we discuss the main modes of operation, including boot-up procedures and FPGA programming. Finally, in Section 6 we summarize the capabilities of RPM and conclude.

2. ARCHITECTURE OF RPM

RPM consists of nine identical boards connected by a Futurebus+ backplane. Eight boards implement the processing elements (PEs) of the multiprocessor and one board is dedicated to I/O. The architecture of each board of RPM is shown in Fig. 1. All processors are LSI Logic L64831 SPARC IU/FPU. These single-chip processors have no on-chip cache and therefore all instruction fetches and data accesses are visible on the pins of the chip.

FIGURE 1. Block Diagram of an RPM Processor Node (I/O board depicted)

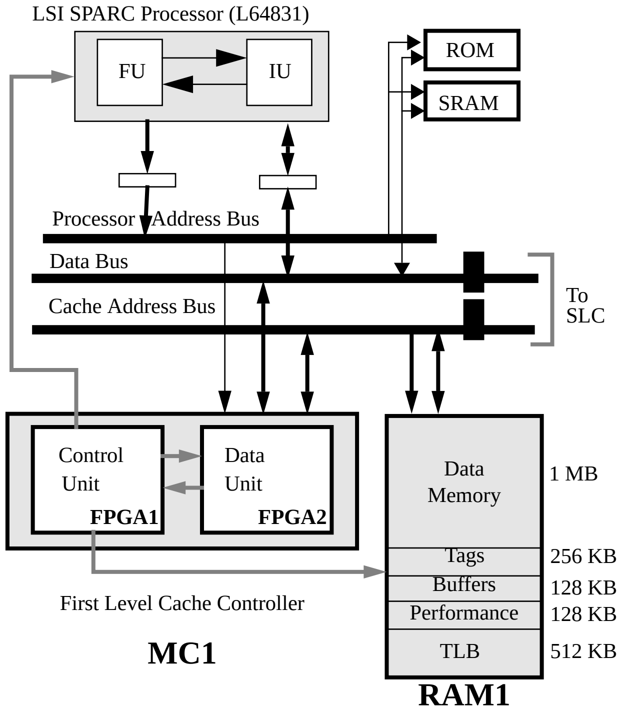


The system is clocked at 10 MHz. This low clock rate allows more complex designs to be mapped into the FPGAs, and facilitates the building of RPM itself. To further increase RPM's capability to emulate complex mechanisms while keeping its cost low, we use multiple system clock cycles to emulate one processor clock cycle (or *pclock*).The number of system clocks per pclock is currently set to eight, giving RPM a peak emulation rate of 10 MIPS (i.e., eight processors at 1.25 MIPS each) or 1.25 million cycles of the target per second.

The I/O board is connected to the host (a Sun SPARC Station II workstation). This workstation serves as the console for RPM and executes its I/O requests. The peak I/O bandwidth is 1.25 MBytes per second, which is more than sufficient for a 10 MIPS machine. Each processor board also has a serial I/O port (RS232) for stand-alone operation and hardware debugging.

There are three levels in the memory hierarchy of RPM, each controlled by a set of Xilinx FPGAs. The first-level memory is tightly coupled with the processor operation and typically implements a first-level cache (see Fig. 2). It consists of 2 MBytes of static RAM (RAM1) controlled by two Xilinx XC4013 (MC1). One FPGA is the control unit (FPGA1) and the other one is the data unit (FPGA2). MC1 paces the execution by blocking the processor at the beginning of each access, executing the sequence of steps needed to satisfy the access, and unblocking the processor when the access is completed. Besides emulating a first-level cache, MC1/RAM1 is responsible for address translation and for the implementation of instruction set extensions (such as prefetch instructions). New instructions can be created by using different Alternate Space Identifiers (ASI) or unused address bits. In our first target emulation, MC1/RAM1 emulates a direct-mapped write-through first-level cache with a block size of 16 Bytes.

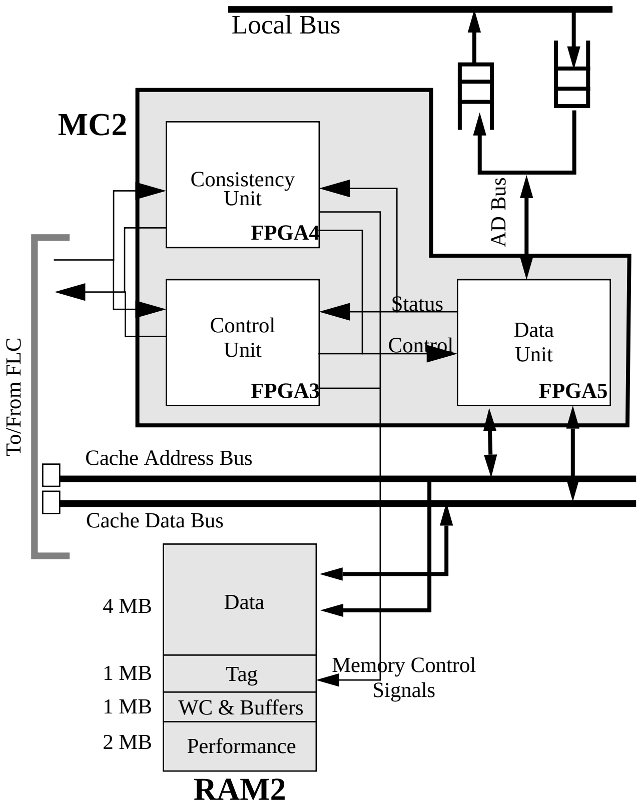
FIGURE 2. The Processor and its First-Level Cache



As a first-level cache (FLC) memory RAM1 is divided into five parts (see Fig. 2): the data memory (up to 1 Mbytes), the cache directory, the TLB (Translation Lookaside Buffer), which is part of the hardware for virtual-memory support, the space for the emulation of prefetch and write buffers, and the space dedicated to the collection of performance statistics (called *count memory*).

The second level in the memory hierarchy is implemented by MC2 (3 Xilinx XC4013) and RAM2 (8 MBytes of static RAM). MC2 connects the processor and MC1 to the rest of the system through the local (on-board) bus. It usually acts as a second-level cache controller. Currently it emulates a two-way set-associative write-back cache with a block size of 16 Bytes (see Fig. 3).

FIGURE 3. The Second-Level Cache Architecture.

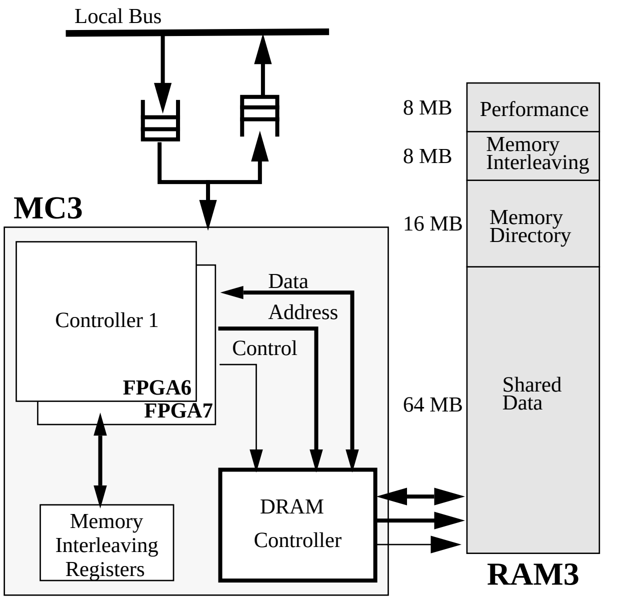


In our current emulation of a cache-coherent system, half of RAM2 is reserved for the second-level cache data memory. The other half is dedicated to the cache directory, various buffers (second-level write buffer and write cache) and count memory. The second-level cache controller is by far the most complex controller of the machine. The data unit (FPGA5) is the data path for the controller, the control unit (FPGA3) implements the basic cache control mechanisms and the consistency unit (FPGA4) contains the control for the mechanisms enforcing access ordering and for the write buffers and prefetching hardware [3]. Since the first emulated target architecture is a sequentially consistent system, the consistency unit and the buffers are not used.

The third level in the memory hierarchy will typically emulate the system main memory and consists of 96 Mbytes of dynamic RAM (RAM3) controlled by two Xilinx XC4013 and one Cypress CYM7232 DRAM controller (MC3). The current emulator implements a directory-based cache protocol and 16 MBytes of RAM3 are reserved for directory entries (see Fig. 4). Some memory is also reserved for performance counters and for the emulation of interleaved memory banks, a mechanism called *virtual interleaving* in [2]. Additional hardware support for virtual interleaving includes eight count down programmable counters implemented with MACH PLDs.

The local bus is a 32-bit wide synchronous packet-switched bus, with a protocol similar to Sun Microsystems MBUS [3]. It connects MC2, MC3 and the Futurebus+ interface.

FIGURE 4. The Main Memory and its Controllers



The Delay Unit (DU in Fig. 1) is a programmable unit which emulates variable interconnection delays. It is a FIFO controlled by one AMD MACH 210 chip. It basically delays the sending of messages to the Futurebus+ so that the expected latency of messages in the target system is respected.

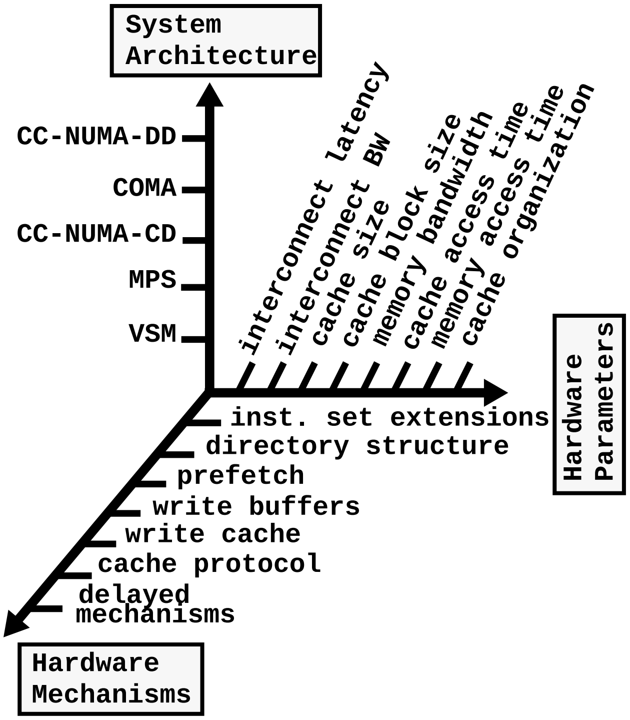
The Futurebus+ interface is made of off-the-shelf chip sets from Newbridge and National Semiconductors. It includes bus transceivers, plus a bus controller (Newbridge’s LIFE chip) and a distributed arbiter from National Semiconductors. The Futurebus+ is used as a point-to-point message passing network.

RPM can emulate FIFO (First-In-First-Out) interconnections with uniform latencies, such as buses and crossbars. Other types of interconnections such as rings can be emulated approximately.

3. EMULATION OF MULTIPROCESSOR ARCHITECTURES

Given its generic board architecture, RPM is capable of emulating in full detail complex target multiprocessors with very different architectures. The flexibility of RPM is captured by the emulation space spanning the possible emulations and illustrated in Fig. 5. Every point in this three dimensional space is a possible emulation of a target system. RPM can emulate different *system architectures* such as Cache Coherent Non-Uniform Memory Access multiprocessors (CC-NUMAs) with centralized (CD) or distributed (DD) directories, Cache-Only Memory Architectures (COMAs), Message Passing Systems (MPS) and Virtual Shared Memory (VSM) systems. Each system architecture has multiple variants with particular *hardware mechanisms* and most *hardware parameters* can be varied within limits.

FIGURE 5. Emulation Space of RPM



The *hardware parameters* are the easiest to modify, either by assigning a different value to a configuration register or by changing a constant in the VHDL file and resynthesizing/re-mapping the design, which can take a few hours. *Hardware mechanism* changes require more effort and may take from a day to a couple of months to finish. Hardware mechanism changes within a given system architecture are facilitated by the fact that significant design reuse is possible. Designing an emulator for a new system architecture is a major task which may take from a month to a few months. We intend to develop a library of system architecture designs for a variety of hardware mechanisms, which will enable us in the future to configure RPM virtually instantaneously, by simply downloading the FPGA configuration from the design database.

Besides emulating a target design at the functional level, RPM can also correctly emulate the timing relationships of the different components of a target system through a technique called *time scaling*. Time scaling is based on the following observation: if the latencies of all components are expressed in terms of pclocks and if all component bandwidths are expressed in terms of bytes per pclock, then functionally identical systems with components of equal latencies and bandwidths are equivalent. Therefore execution times measured in pclocks on RPM are equal to execution times in pclock on the target, provided latencies and bandwidths of RPM's components are appropriately adjusted. For more detailed information please consult [2].

4. DESIGNING NEW EMULATIONS

As explained in Section 2, seven FPGAs implement the controllers of the caches and the main memory in RPM. Different target systems can be emulated by programming these FPGAs. In the case of cache-coherent systems, which we are currently investigating, the design process goes through the following phases. At first, a high-level state diagram is drawn for each controller (*states* here refer to cache or directory states and not to states of finite-state machines). Next, this diagram is refined into state tables which specify the protocol at a high abstraction level. Flowcharts are then derived for every possible request a controller may receive. From the flowcharts the control logic and the data paths are designed and then translated into VHDL code, which is synthesized and mapped into the FPGAs. Simulations currently rely on Viewlogic's Viewsim tool.

Table 1: FPGA Statistics

Controller	FPGA No	Occupied CLBs (%)	Packed CLBs (%)	# of IO pins used
MC1	1	73	47	122
	2	60	41	176
MC2	3	82	54	118
	4	not used	not used	not used
	5	95	73	171
MC3	6	89	65	141
	7	66	47	95

Initially, Viewlogic's Viewsynthesis 2.2.1 was our synthesis tool, but as the complexity of the controllers increased Viewsynthesis 2.2.1 was producing inefficient designs. To remedy this problem we developed an interface tool which converts the intermediate file generated by Viewsynthesis into bliff format, the input format for Berkeley's SIS tools. Our designs are now synthesized by SIS and the synthesized designs are mapped onto the FPGAs with Xilinx's XACT 5.0 tools.

Table 1 shows the utilization of the FPGAs for our first emulated target system (a sequentially consistent CC-NUMA system). The values in Table 1 were obtained without significant optimization. We believe that they can be improved with further optimization and better synthesis tools. We also plan to replace our XC4013 FPGAs with pin-compatible XC4025s as soon as they become available. This will nearly double the number of gates in each FPGA.

The FPGAs within each controller are connected to each other through most of their unused pins in order to accommodate additional signals between FPGAs in future designs. Some pins are reserved for global signals connected to all FPGAs. Others are used to probe signals inside the FPGA for debugging purposes.

Our choice of VHDL to program the FPGAs has brought several benefits. The use of VHDL reduces the design time, facilitates maintenance and debugging, and provides an excellent documentation of the design. Keeping some parameters of the design such as cache block size and cache size as VHDL constants simplifies some common design changes.

The use of VHDL and CAD tools, especially synthesis tools, and the services of EZFAB offered by ISI for the design of the boards made it possible to complete the design and the construction of RPM in just 15 months by five graduate research assistants. Currently the partition among FPGAs of each controller design and the signal assignments are done manually with Viewdraw, the Viewlogic schematic entry tool. A tool which automatically partitions circuit designs and which assigns signals while obeying some constraints (such as committed I/O pins) would significantly cut the design time of new emulations. Moreover, very high-level synthesis tools which accept an algorithmic circuit description would also speed up the design process.

5. OPERATION OF RPM

RPM normally operates as a slave to a host workstation serving the role of console and from which FPGA programs and pre-compiled SPARC binaries are downloaded. However it can also operate in stand-alone mode (mainly for debugging purposes), using default code and FPGA programs contained in its boot ROMs.

Regardless of the target system being emulated, some basic features are always present. These features include the capability to read/write all memories in the system from any board, inter-processor interrupts, software system reset, a backplane hardware barrier synchronization, and the ability of the I/O board to reset and reload any or all FPGAs in the system. Additionally, all the main finite-state machines have a special protocol error handling procedure. Whenever a condition is found to violate the protocol specifications in a controller, relevant state information is saved in well known locations in its associated memory and the processor is interrupted. This feature greatly facilitates the debugging of complex coherence and communication protocols.

Also included in all FPGA programs is the default performance monitoring mechanism, called *count memory*. For each event activating a controller a unique address is generated. This address points to an event counter in the area of memory allocated to performance counters. The controller fetches, increments, and stores back the counter. Since each pclock takes multiple system clock cycles, there are enough idle cycles in all controllers to make this mechanism completely non-intrusive.

5.1 Boot-up Procedure

After the power-up or reset the boot-up procedure is executed by all boards. Boot-up procedures and a basic set of FPGA programs are stored in the boot ROM of every board. During the boot-up phase, instructions and data are fetched from the boot ROM and the boot SRAM is used for storing dynamic data. The boot-up procedure proceeds as follows.

First every board performs a self-test, halting its operation if the test fails. The self-test checks the boot ROM checksum, the boot SRAM, and the SCSI and RS-232 controllers. After the self-test all boards initialize their DRAM controllers. The PEs wait for the I/O board to program their FPGAs. The I/O board can download either the FPGA programs contained in the boot ROM or a new set of programs from the host through the SCSI interface. Every PE can find out whether the programming of their FPGAs is completed by sampling two global signals (see Section 5.2).

Once the FPGA programs have been downloaded each board reads its own *boardID* register from MC1 and enters the *initialization mode*. In this mode, MC1 responds only to data accesses from special ASI load/store operations called *test mode accesses*. Normal data and instruction accesses are still satisfied by the boot ROM or boot SRAM. Using these test mode accesses one processor can read/write any memory on any board and initialize all relevant registers.

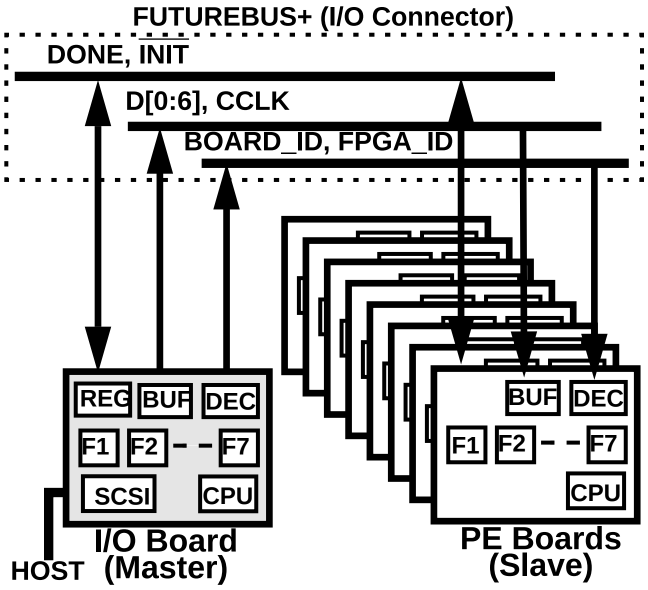
At this point, the I/O board transfers the binary image of the software program from the host into the main memory of one or multiple boards, while the other boards spin-wait. When the downloading is complete, all PEs clear their RAM1 and RAM2 banks and execute a global barrier. After all boards have reached the barrier every board switches to *emulation mode* and starts executing the program. The switch to emulation mode is done by clearing a register with a test mode store and by jumping to the starting point of the program. By clearing the register, the boot ROM and

SRAM are removed from the addressing space and MC1 is signalled to start responding to instruction fetches and normal data accesses as well as test mode accesses. As soon as all boards have switched to emulation mode the emulation of the target architecture begins.

The program loaded by the I/O board has been linked with all required libraries as well as a basic run-time system. The program contains start/stop macros to control the mechanism for performance statistic collection. After the execution of the program is finished, every PE waits while the I/O board uploads the results and the collected performance statistics to the host computer.

The SCSI protocol between the I/O board and the host workstation is master/slave, the workstation being the master. In other words, only the host workstation can initiate SCSI transactions. To allow RPM to initiate the communication when necessary an extra RS-232 connection has been added between the host and the I/O board.

FIGURE 6. FPGA Programming Logic



RPM can also operate in stand-alone or *debugging mode*. In this mode, instead of being connected to a host workstation, RPM is connected to an ASCII terminal through one of the RS-232 ports of the I/O board. The operation in debugging mode is similar to the above; however the FPGA programs and SPARC binaries are loaded from the boot ROM of the I/O board. This mode is mainly used for debugging the correctness of an emulator. In this case, the system can also switch to emulation mode, but it is only capable of executing a variety of tests selectable from the ASCII terminal console.

5.2 FPGA Programming

FPGA programming is done by the I/O board through unused Futurebus+ I/O backplane connector pins (see Fig. 6). The bus lines connected to these pins are used as bare wire connections between the I/O board (master) and the PEs (slaves).

The I/O board uses two registers to select which FPGAs it will program. One is the FPGA Selection register and the other one is the FPGA data/clock register. The configuration of the FPGA selection register is shown in Table 2. FPGA_NO can be from 1 to 7 and BOARD_NO can be from 1 to 9. All zeros means no selection. 15 is used to select all boards or all FPGAs or both. 14 is used to reset all FPGAs on the selected board. The Xilinx FPGAs are configured in Slave Serial Mode and the I/O board supplies both the clock (CCLK) and the bit serial data for all FPGAs. The configuration data is latched into the FPGA Data/Clock register (Table 2) and configuration clock CCLK is generated by first resetting bit 7 of the FPGA Data/Clock register and then setting it. A single XC4013 requires 247,960 bits (~30KBytes) of configuration data and the time required to program the seven FPGAs of all boards is approximately one second.

Table 2: FPGA Selection Register

D7	D6	D5	D4	D3	D2	D1	D0
FPGA_NO				BOARD_NO			

Table 3: FPGA Data/Clock Register

D7	D6	D5	D4	D3	D2	D1	D0
CCLK	Serial Data per each FPGA						

During the programming of the FPGAs, the detection of a frame error aborts the procedure by pulling down the INIT line. If there are no frame errors, the DONE signal goes to high when the FPGA programming is finished. Activation of all FPGAs on all boards is synchronized with the DONE signal and their corresponding system clocks. All boards can spin on the values of INIT and DONE through special load instructions.

6. CONCLUSION

Current trends show that small-scale multiprocessor systems will be common place in the near future. In particular, shared-memory multiprocessors with hardware cache-coherence are very complex systems, for which the design times tend to be relatively long with respect to uniprocessor systems.

To make such systems competitive it is necessary to learn more about the effectiveness of different multiprocessor architectures under different workloads to determine the most promising designs, under the shortest possible design cycle. Simulators have been widely used for this purpose. As the complexity of the examined systems increases and as the data set sizes of the typical application programs grow, it becomes more and more difficult to simulate these systems in detail. Architecture simulators which typically model a system at a very abstract level may take several hours to simulate a second of the execution of the target system, while detailed cycle-by-cycle simulators are usually much slower.

RPM is an unique tool that bridges the gap between simulation and prototyping of multiprocessor systems through hardware emulation. RPM has a slowdown factor in the order of hundreds whereas simulators have a slowdown factor of tens of thousands with respect to a target system. Since every emulation is an actual incarnation of the target system, RPM emulations are much more detailed than typical simulation models, providing valuable insight into the design of the target system.

RPM utilizes FPGAs in the controllers at all levels of the memory hierarchy and uses off-the shelf components for the rest of the system. This feature makes it possible for RPM to emulate various multiprocessor systems, including CC-NUMA architectures, COMAs, Message-Passing Systems, and Virtual Shared Memory Systems. Many different hardware mechanisms for these systems can also be implemented. Moreover, RPM is able to run realistic workloads such as operating system kernels, database engines, and multitasked scientific workloads with the real data set sizes for which the target machine is built.

7. BIBLIOGRAPHY

[1] Aptix Inc., *Data Book*, Aptix, San Jose, 1993.

[2] Barroso, L., Iman, S., Jeong, J., Öner, K., Ramamurthy, K., and Dubois, M., “RPM: A Rapid Prototyping Engine for Multiprocessor Systems”, *IEEE Computer*, February 1995.

[3] Catanzaro, B., *Multiprocessor System Architectures*. Prentice-Hall, 1994.

[4] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta, “SPLASH: Stanford parallel applications for shared-memory”, Technical Report CSL-TR-91-469, Stanford Computer Systems Laboratory, April 1991.

[5] Dubois, M., Scheurich, C., and Briggs, F.A., “Synchronization, Coherence, and Event Ordering in Multiprocessors,” *IEEE Computer*, Vol. 21, No. 2, pp. 9-21, February 1988.

[6] Hagersten, E., Landin, A., and Haridi, S., “DDM -- A Cache-Only Memory Architecture,” I*EEE Computer*, Vol. 25, No. 9, pp. 44-54, September 1992.

[7] INCA, “VA-II Logic Emulator”, INCA, Berkshire, United Kingdom, 1993.

[8] Ko, B. and Maholtra, L., “Logic Emulation for System-Level Design”, *Proc. Electro*, 1992.

[9] Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W.-D., Gupta, A., Hennessy, J., Horowitz, M., and Lam, M.S., “The Stanford DASH Multiprocessor,” *IEEE Computer*, pp. 63-79, Vol. 25, No. 3, March 1992.

[10] Li, K. and Hudak, P., “Memory Coherence in Shared Virtual Memory Systems,” *ACM Transactions on Computer Systems*, Vol. 7, No. 4, pp. 321-359, November 1989.

[11] Maliniak, L., “Logic Emulation Meets the Demands of CPU Designers”, *Electronic Design*, vol. 41, no. 7, pp. 36-40, 1993.

[12] Stenstrom, P.,”A Survey of Cache Coherence Schemes for Multiprocessors,” *IEEE Computer*, Vol. 23, No. 6, pp. 12-24, June 1990.

[13] Trimberger, S.,”A Reprogrammable Gate Array and Applications,” *Proceedings of the IEEE*, Vol. 81, No. 7, pp. 1030-1041, July 1993.

[14] Walters, S., “Reprogrammable Hardware Emulation Automates System Level ASIC Validation,” *Proc. WESCON*, 199011