A Chip Prototyping Substrate: The Flexible Architecture for Simulation and Testing (FAST)

John D. Davis, Stephen E. Richardson, Charis Charitsis, Kunle Olukotun
Computer Systems Lab
Stanford University
(johnd, steveri, charis, kunle)@ogun.Stanford.edu

Abstract

We describe a hybrid hardware emulation environment: the Flexible Architecture for Simulation and Testing (FAST). FAST integrates field-programmable gate arrays (FPGAs), microprocessors, and memory to enable rapid prototyping of chip multiprocessors, multithreaded architectures, or other novel computer architectures and chip-level memory systems. FAST combines configurable and fixed-function hardware and software to facilitate rapid prototyping by utilizing components optimized for their particular tasks: FPGAs for interconnect and glue logic; processors for rapid program execution; and SRAMs for fast memory. Unlike software simulators, FAST can simulate complex designs at multi-megahertz speeds regardless of the simulation detail. We illustrate FAST's utility by describing mappings of both a small-scale CMP with speculation support and a large-scale CMP connected using a network. We then show performance results from a very simple, decoupled 4-way CMP executing small test programs.

1. Introduction

Multithreaded microprocessor architectures such as chip multiprocessors (CMPs) are now ubiquitous in industry and research. Traditionally, computer architects have leveraged increasing transistor density on microprocessor chips to implement single, large processors that exploit instruction level parallelism (ILP). However, continued performance gains from ILP are becoming increasingly difficult to achieve due to limited parallelism among instructions in typical applications [26]. Likewise, the problems associated with designing ever-larger and more complex monolithic processor cores are becoming increasingly significant. These problems include higher bug rates, longer design and verification times caused by the design complexity and the need to design for increasing wire delay [19]. This reality has spurred great interest in exploiting thread-level parallelism (TLP) among independent threads of instructions to continue historical microprocessor performance improvement trends. These multithreaded architectures effectively integrate multiple homogeneous or heterogeneous processors onto a single chip [5,16].

Researchers have shown that chip-level multithreaded microprocessors enable many new TLP extraction techniques, which conventional symmetric multiprocessors (SMPs) cannot exploit, by providing an order of magnitude improvement in interprocessor communication latency and bandwidth compared to conventional SMPs. These improvements in communication performance enable conventional parallel applications to be divided up into very fine-grain threads that can achieve parallel speedups even when they would slow down on conventional SMP hardware. Using thread-level speculation (TLS) support, even nominally sequential applications can be broken into collections of fine-grain threads that can be run in parallel on CMPs. TLS support adds hardware to guarantee that the sequential application will still execute correctly in parallel. It does this by tracking dependencies and backing up any threads that violate the original program's dataflow [8,15,23,24]. By eliminating the need for guaranteed thread independence, TLS makes it much simpler to parallelize sequential applications.

Unfortunately, because very few actual processor implementations that are available exploit fine-grain TLP, and because no TLS architectures exist, researchers have been forced to rely on simulation or emulation to evaluate performance and develop software techniques to exploit CMPs [1,7,10,17,20,21,25,35,37]. Software-based simulators allow one to evaluate small benchmarks or fragments of larger benchmarks using instruction-level simulation, but are too slow to simulate entire applications within a reasonable time. Complicated CMP and multiprocessor designs exacerbate this problem by requiring that many processors be simulated simultaneously [5,8,11,15,18,22-24]. The complexity of even the most basic multithreaded architectures limits instruction-level simulation to an effective "clock rate" of about 0.05 MHz; most simulators, especially RTL ones, achieve much less [1,7,10,17,21,35]. Simulation speed therefore limits the scope and effectiveness of research that can be performed in reasonable amounts of time.

Many efforts have been made to overcome software speed limitations using hardware emulation [20,28,30]. Historically, hardware emulation platforms using arrays of FPGAs have been used to generate rapid prototyping systems that can simulate entire applications at an RTL level [2,6,25]. Unfortunately, efforts to compile multiprocessor designs to these systems have been limited by poor FPGA logic utilization, limited interconnectivity in the FPGA arrays, and poor word-size data manipulation by bit-width FPGA logic units [25].

To address these problems, we have built a Flexible Architecture for Simulation and Testing (FAST). FAST uses simple processors combined with state-of-the-art field-programmable gate arrays (FPGAs) and memory chips on a single printed circuit board (PCB) to create a flexible simulation fabric that can execute millions of instructions per second. The resulting environment executes code at speeds at least two orders of magnitude faster than execution-driven software simulation and an order of magnitude faster than previous hardware emulation using generic arrays of FPGAs [2,6]. Using FAST, researchers can rapidly prototype a variety of CMP architectures in a relatively short amount of time. FAST allows detailed investigation of many topics related to the design of CMPs, including overall system design, memory hierarchy structures, TLP extraction methods, TLP-oriented software design for operating systems and high-level applications, reconfigurable architectures, embedded systems, and ISA extensions.

This paper presents the initial implementation of our configurable hardware emulator. Section 2 presents an overview of the system. Section 3 provides a few motivating examples of how CMP architectures can be mapped to the FAST PCB for emulation. Related work is evaluated in Section 4, while our conclusions and future plans are presented in Sections 5 and 6.

2. FAST Details

The FAST system is a collection of hardware and software components that manage and configure on-board resources. As shown in Figure 1, these resources exist as functional layers that together can simulate multiprocessor hardware systems at high speeds. The layers include: several fixed-function memories and microprocessors (Hardware), FPGA devices that can be "morphed" to provide different system-level functionality using a variety of Verilog memory hierarchy models (Morphware), and application benchmarks to be evaluated, low-level software and a batch operating system to manage functions such as program loading and I/O (Software). These are depicted from the bottom up in Figure 1.

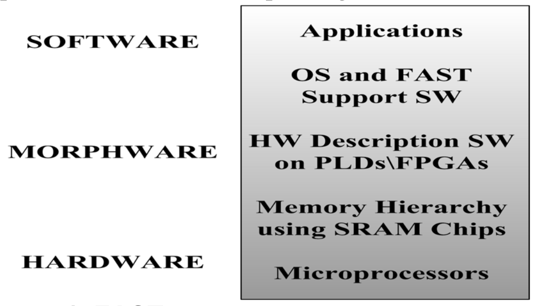
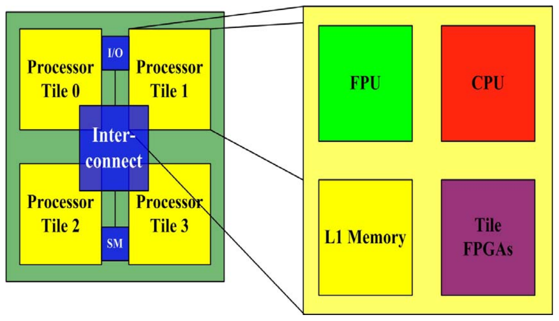


Figure 1. FAST system components.

2.1. FAST Hardware



FAST Overview	Quantity
Level 1 (L1) Memory	1 MB/tile
Level 2 Memory (SM)	64 MB
Flash Memory	16 MB
Integer ALU (CPU)	1/tile
Floating-Point ALU (FPU)	1/tile
Tile FPGAs	2/tile
Board FPGAs (Interconnect)	2/board

Figure 2. FAST high-level overview.

FAST is a single printed circuit board (PCB) system comprised of four replicated processor tiles and an associated top-level interconnect, as shown in Figure 2. Each processor tile contains: an integer and a floating point datapath, 1 MB of processor local memory, and FPGAs that manage the processor tile resources and facilitate reconfiguration. The top-level interconnect is composed of larger FPGAs to allow communication between the four processor tiles. It provides several shared resources, including on/off-PCB I/O via

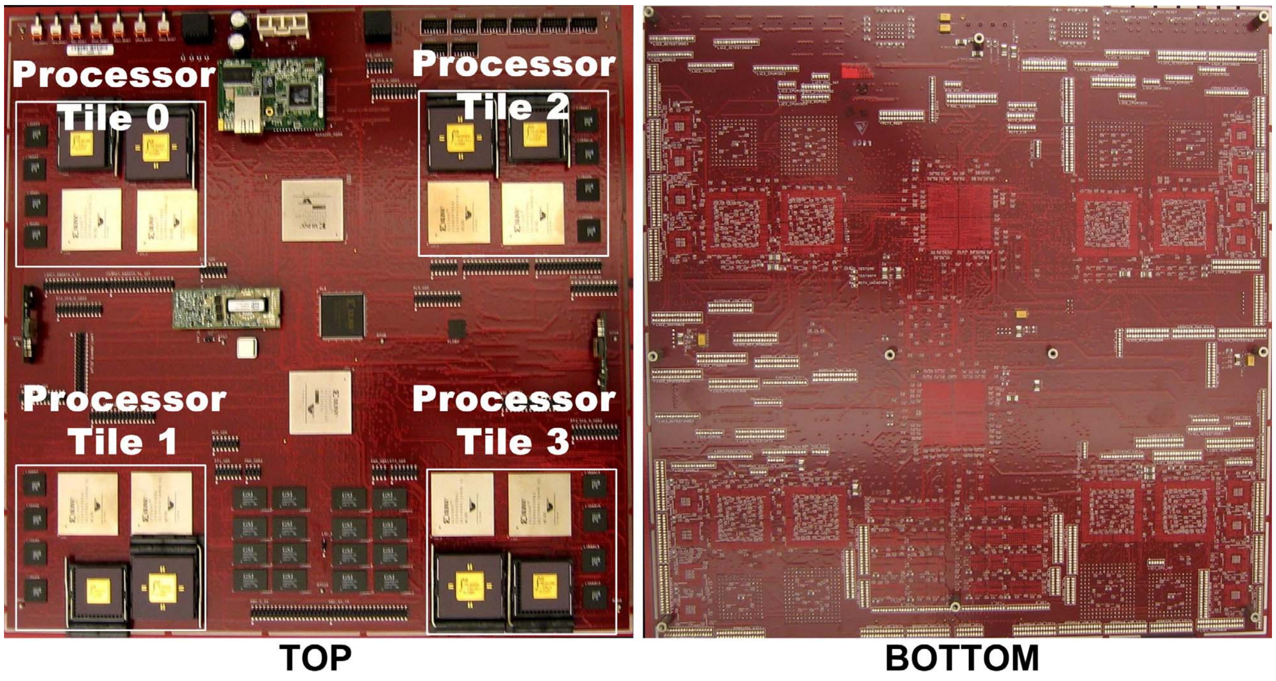


Figure 3. The FAST PCB.

Ethernet and TCP/IP, an expansion connector, 64 MB of second level memory, on-board Flash memory, and hardware to manage these resources and facilitate reconfiguration.

Table 1. Component operating frequency and core voltage.

Component	Quantity	Frequency Max. (MHz)	Core Voltage (V)
XC2V6000 FPGA [38]	2	400	1.5
XCV1000 FPGA [36]	8	100	2.5
Primary Cache SRAM [32]	16	100	3.3
Secondary Memory SRAM [31]	16	200	3.3
Ethernet Module [33]	1	44	3.3
CPLD [38]	1	100	3.3
Flash [27]	1	10	3.3
MIPS R30X0 CPU, FPU [25]	8	25	5

Figure 3 is a photograph of the actual FAST PCB. The 20-layer board is 16" x 16". There are over 27,700 vias with a total of 43 BGA parts and approximately 4220 surface mount components, with about half used for test points to access various hidden FPGA ball grid array bumps. Processor tiles occupy each corner of the PCB. The interconnection components, including board management, off-PCB communication, and secondary memory, occupy the center of the PCB. Table 1 summarizes the FAST PCB components, their operating frequency, and core voltage.

The MIPS R3000 CPUs and R3010 FPUs were chosen to form the processing core of the system for several important reasons. First, they provide an exposed primary cache and coprocessor interface, unlike all present-day processors and commercially available "hardcore" processor macros in FPGAs. This architecture allows us to observe or modify the primary cache interface to the CPU as necessary. Second, the presence of the FPU expands the application domain to include floating-point intensive applications, which would be impractical to run with slow floating-point emulation. Third, using a "hard" processor core instead of a "soft" FPGA-implemented core leverages the hard core's highly optimized datapath, which is better suited for word-size data manipulation than groups of FPGA logic blocks. Furthermore, the simple R3000 pipeline suits FAST's goal of exploring non-ILP-intensive architecture innovations, such as TLP in simple CMPs, and novel memory hierarchies. However, even though our main goal is memory hierarchy and TLP exploration, Section 3.3 explains how simple cores can emulate more complex cores. Of course, the use of hard MIPS cores does limit FAST to MIPS ISA, but various tricks can be used, such as binary translation or soft cores, to expand FAST's ISA options.

The "simulator" clock speed is set by the processors' 25 MHz clock frequency. The FPGAs and SRAM memories can operate at much higher speeds enabling time-division multiplexing of hardware resources. This allows us to emulate larger virtual bus widths, multi-way caches, or other resources. Thus, we are able to achieve a maximum of 100 MIPS on our four-processor system with simple, in-order processors. This is an order of magnitude less than currently available microprocessor peak operating performance. However, it compares very favorably against all forms of software-based simulation. Using full-system execution-driven simulators of very simple instruction-level models on high-speed microprocessors, one can achieve only 1-10 MIPS. With more complex instruction-level models, or especially with RTL-level models, the effective rate is reduced to levels of 0.05 MIPS or less using the same high-speed microprocessors. Taking memory delay inefficiencies into account, we expect FAST to average about 25–50 MIPS for CMP full-system simulations. Therefore, our system should realistically provide a performance gain of nearly three orders of magnitude over equivalent, software-only simulators.

The FAST PCB presented many hardware design challenges, including a mixed voltage design environment, several clocking domains, and a complex combination of state-of-the-art components with outdated processors. We chose components that provided the best performance alternatives while satisfying our pin-level visibility constraints. In order to manage the potential design complexity, we selected parts to tolerate the variations in technologies on the PCB. There are four different voltage domains on the board, with core device voltages ranging from 1.5V to 5V. For simplicity, however, essentially all I/O operates at 3.3V. The only exceptions to this rule are the 5V outputs from the processors. To avoid voltage level shifters required to interface 5V signals to the modern FPGAs at the heart of the board, we integrate two generations of FPGAs, reducing FAST's cost and complexity. As shown in Figure 4, a summary of interconnections within the system, only the older, 5V-tolerant XCV1000 parts interface directly to the MIPS components.

2.2. FAST Morphware

The FAST morphware layer describes the field-programmable gate arrays (FPGAs) and the Verilog modules that enable explicit memory hierarchy manipulation and investigation. Our state-of-the-art FPGAs form a versatile hybrid hardware emulation platform. In aggregate, there are over 20 million programmable system gates on the FAST board. While most of these gates will be used to configure the memory hierarchy, many can also implement embedded counters and other system monitors to collect detailed execution statistics at full speed. Thus, we have a highly detailed full-system simulator without the slowdown associated with equivalently detailed software simulators.

To fully utilize the FAST prototyping substrate, one must compile a Verilog model of the memory system, performance counters, and associated components. The main FAST-specific part of this task is to partition the target architecture, or at least a portion of it, across the various FPGAs. In general, the modern FPGAs at the heart of FAST provide copious amounts of on-chip bandwidth, but the number of pins on an FPGA limits inter-chip bandwidth and therefore may limit FAST's ability to emulate some architectures. However, the high FPGA clock speeds relative to the 25 MHz "system" clock do allow some time-division multiplexing of the inter-chip buses, enabling virtual bus widths much larger than physical resources. Figure 4 illustrates the FPGAs and the physical bus widths between the components on the PCB. We have developed a basic Verilog wrapper that instantiates all of these point-to-point connections, to act as a "shell" for any suitably partitioned design. With this shell, the fixed parts of the FAST board hardware can be used across multiple designs.

The remainder of this section describes how a typical architecture will use the various flexible components to implement a wide variety of designs. While some typical memory system functions could be placed in one or more of the FPGAs, the jobs assigned to any particular chip are somewhat limited by the fixed hardware resources attached to it.

The CPLD (XCRL3512XL) and associated Flash memory (right side of Figure 4) are the system bootstrapping and monitoring devices. The CPLD programs the FPGAs using the Xilinx 8-bit parallel programming ports. There are 8 JTAG groups on the PCB as well, to provide a secondary system programming and debugging port. The CPLD also acts as the memory controller for the Flash memory chip. This 16M x 8 bit Flash chip (AMD AM29LV652D) has the capacity to hold all 10 FPGA configurations with at least 5 MB remaining for storing bootstrap software code right on the board. The CPLD also controls the overall state of the board, choosing among FPGA configuration, application execution, application debugging, and reset modes.

Each central control FPGA provides over 1100 I/O pins, a maximum operating clock frequency of 400 MHz, and 6 million system gates each. The Read/Write Controller (RWC) handles memory hierarchy events that propagate past the primary caches, such as

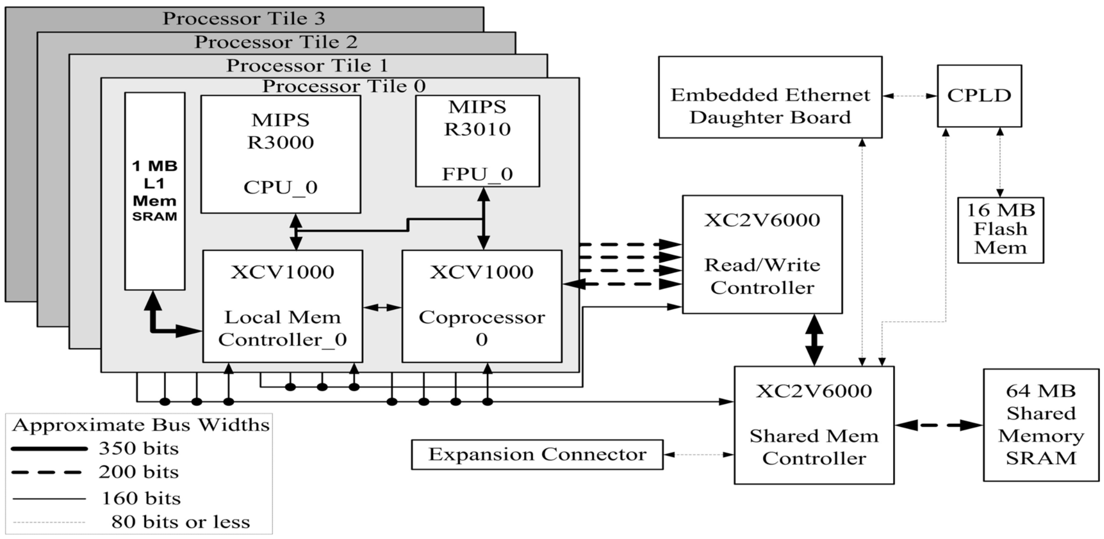


Figure 4. FAST functional diagram with major buses.

write-throughs and cache misses. A wide bus permits the observation of memory traffic to and from all processor tiles on a cycle-by-cycle basis, making it possible to implement cache coherence protocols requiring snooping on primary cache contents in other processor tiles. This controller could also be used for inter-processor tile messaging in systems that do not use traditional memory coherence [4]. Meanwhile, the Shared Memory Controller (SMC) manages the 16M x 36 bit (including 4 ECC bits) secondary memory. These synchronous SRAMs can be configured as a secondary cache or general-purpose, off-chip memory. The entire memory can be shared by all processor tiles or segmented into 4M x 36 bit private partitions assigned to each tile. Furthermore, time-division multiplexing can be utilized to implement various set associative cache configurations. The SMC also controls an 80-pin expansion connector. This multi-purpose header can be used to connect multiple FAST prototyping substrates together to create a larger FAST emulation fabric, to add daughter cards, or to attach additional memory, such as a DRAM main memory bank.

Each processor tile retains some flexibility with its two Xilinx XCV1000 FPGAs. One FPGA is assigned to be the Local Memory Controller (LMC), and manages the 256K x 36 bit (including 4 ECC bits), dual-ported local processor tile memory. The LMC serves as the R3000's external cache interface, enabling a wide range of virtual cache configurations while the morphware fakes the expected direct-mapped cache behavior. Like the SMC, the LMC can utilize time-division multiplexing to implement set associative caches or other types of memory systems. Furthermore, the CPU and FPU always access the local memory through the LMC, giving it the ability to modify the instruction and data stream on-the-fly, if necessary. The second XCV1000 is a "Coprocessor FPGA" that can be used to add instructions to the MIPS ISA or to maintain statistics counters. The MIPS-I ISA has a well-defined coprocessor interface that can be exploited by this FPGA to add instructions to implement software control over cache coherence protocols, additional functional units, or other features.

2.3. FAST Software

There are several software components that facilitate FAST's hardware emulation, including: applications, a program loader, an interactive debugger, a low-level operating system on the FAST board itself, and a monitoring and OS support environment that runs on an attached SGI host workstation.

FAST uses the MIPS-I ISA and the related MIPS-SDE toolchain running on SGI workstations for application development. MIPS-based SGI machines simplify the building and testing process for the MIPS processors on FAST, but a host running a non-MIPS ISA is also possible. Using the MIPS-SDE toolchain, we create a statically linked ELF binary. By using only statically linked binaries, we eliminate the need for dynamic linking support in the program loader and/or on-board OS.

Once generated, a binary can be loaded on the FAST simulation substrate using one of two different program loader modes that establish a TCP/IP socket connection with FAST to download the binary image into FAST's secondary memory. The first loader mode provides an interactive interface, while the second runs batch scripts non-interactively. The interactive terminal program facilitates application debugging and monitoring through an interface similar to GNU gdb [31], while scripting enables rapid batch execution of binaries to get performance results. For initial board testing, there is also a "backdoor" JTAG interface that can directly load programs into FAST's secondary memory, bypassing the normal TCP/IP-based board I/O.

To provide low-level OS support for applications executing on the board, we are porting the PMON system monitor produced by LSI Logic and Algorithmics. PMON provides application monitoring and debugging for MIPS-based evaluation boards; we are modifying it somewhat to add batch operating system functionality for FAST [31]. PMON provides the gdb-like interface for application debugging that makes FAST's processor state and memory state visible to the user in interactive execution mode. We also use it to provide limited OS support for libc functions when any of the MIPS processors execute SYSCALL instructions. When I/O with the host workstation is necessary, PMON uses Ethernet through a microcontroller on the FAST board that handles the TCP/IP protocol details. The microcontroller transmits messages from the FAST PCB to the host using both simple terminal-style I/O and using special messages when OS functionality requires external support for functions like file I/O. These "OS" messages reduce the complexity of the on-board FAST OS significantly while still enabling simulation of real-world applications that require significant OS support. For example, in the case of file I/O, the on-board OS only needs to communicate file handles and buffers associated with read() and write() calls between the application and host interface, while leaving the details of disk management solely to the host's operating system.

3. Mapping Designs to FAST

The FAST prototyping substrate is designed to map architectures that employ fine-grained threading, speculative threading, and chip multiprocessing. We believe FAST will be able to emulate many examples of unimplemented research architectures that use these features. This section describes HYDRA, Smart Memories, and other specific architectures that could be mapped to the FAST substrate in a reasonably efficient manner.

3.1. A Simple CMP with TLS Support, HYDRA

We initially intend to exploit FAST's prototyping potential to explore architectures that make use of thread-level speculation (TLS). This technique offers the potential to improve sequential program performance by dissecting a conventional sequential program into multiple small threads and attempting to execute these small threads in parallel. This requires hardware support to monitor writes of data by "earlier" threads that may potentially be read by "later" threads executing in parallel. In the event that a "later" thread reads a value too soon, it must be squashed and restarted so it may read the new, correct value. Several researchers have proposed techniques for controlling the speculative threads and maintaining the sequential order of committed program state [8,15,23,24]. Since most of these techniques were designed to work on CMPs, FAST should be able to emulate most of these architectures quite well, with one speculative thread assigned to each of FAST's processor tiles at any given time. Because simulation of these architectures was so important to us, we examined an RTL description of Hydra [8] during the development of the FAST PCB to help guide the inter-FPGA pin allocation on the FAST board. Connections to allow high interprocessor snooping bandwidth were included as a result of these investigations.

To date, no system with TLS has been built. This has forced all TLS evaluation to be performed with software simulation on small test benchmarks or sampled portions of larger benchmarks. Unlike these software simulators, FAST will be able to provide detailed execution statistics for *full* benchmarks running on these architectures. In addition, since FAST has full control over the various system latencies relative to the 25 MHz processor system clocks, FAST allows more extensive evaluation of parameterizable system features, such as memory latencies, than is practical with slow software-based simulation.

The FAST prototyping environment will also facilitate the development of application software and programming environments optimized for speculatively multithreaded architectures. Software simulation is too slow to support the development of the software infrastructure required either for performance tuning of applications that have been manually converted to use TLS or for automated compilation of sequential TLS-enabled programs. As a result, we expect to use FAST's high-speed simulation to gain insight into general techniques that can be used by programmers or compilers to exploit the full potential of TLS-based parallelization.

3.2. A Large-Scale, Networked CMP

Several research projects have proposed architectures composed of multiple "tiled" processors on a single chip [18,22,25]. Since FAST can only simulate four-core systems, at most, one might initially conclude that it is not useful for emulating these architectures. However, FAST can still prove useful with these architectures. Many insights can still be gained by simulating just a four-processor subsection of the design using a single FAST board. Furthermore, one can use the expansion connector to emulate larger systems by connecting multiple FAST boards together. While the expansion connector only has a relatively limited number of pins, most of these architectures use a network that limits the number of long wires needed to connect processors together when they are physically distant from one another on the chip. Such relatively narrow networks should map well to the expansion port. If necessary, some of the secondary memory could also easily be used to support network buffering requirements, if they became too large for on-FPGA buffers.

In order to verify these ideas, we looked closely at the Stanford Smart Memories design [18] while designing the expansion port. This CMP has 10's of processor tiles clustered together into small groups of tiles that share a common network port. A single FAST board could emulate a group of four processors, while the expansion port could be daisy chained to additional FAST boards to allow emulation of a large system.

3.3. Emulating More Complex Cores

The MIPS R3000 external interfaces to both cache and coprocessors provide visibility that can be used to transform FAST's simple in-order single-issue cores into a wide variety of other microarchitectures, because the two FPGAs in each processor tile have full control over the data and instruction streams fed into the processor. Auxiliary structures can be maintained in these FPGAs, such as counters and monitors, to make it possible to change the definition of a "simulated machine cycle" or to adjust or interpret the instruction streams. With these FPGAs, we can define several MIPS R3000 cycles as one target machine cycle to gang instructions together into "single-cycle" packets.

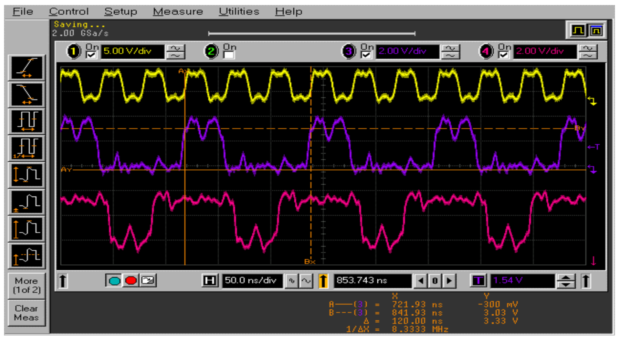
Depending on the underlying architecture, these instruction packets could be executed intra-tile (very long instruction word, *VLIW*) or inter-tile (single instruction multiple data, *SIMD*). We prefer to execute VLIW packets serially, instead of in parallel across the processor tiles, because simulating the shared register file used by all VLIW issue slots across processor tiles would require many extra store instructions to make all instruction results visible to the FPGAs. On the other hand, SIMD packets can be spread across processor tiles executing identical instruction streams because one instruction specifies the operation for several parallel data "lanes" of execution, which do not share data between lanes on a cycle-by-cycle basis. When the number of "lanes" exceeds the number of processor tiles, both serial and parallel execution methods may be used. Finally, cores with more complex instruction fetch mechanisms could also be implemented using these techniques with variable-length "simulated machine cycle" times. Implementing an instruction window in the LMC could be used to enable fine-grain multithreaded emulation and/or wide-issue superscalar core emulation.

3.4. Embedded SOC Architectures

VLSI process scaling has increased the complexity of embedded and application-specific processor-based designs. FAST enables full or partial system emulation for a wide variety of these systems by manipulating the memory hierarchy and defining the number of processor tile cycles per target machine cycle. MIPS-based processor cores are widely used for embedded systems and we envision collections of these processors working in concert for system-on-a-chip (SOC) designs. FAST is an ideal prototyping platform for continued research in these types of embedded systems [20].

4. Initial Performance Results

In FAST's initial stages of bring-up, testing, and prototyping, we have successfully powered-up the PCB, programmed all the FPGAs with simple test Verilog, programmed the coprocessor FPGA to issue instructions, and run small test programs on the R3000s. Initially, we have configured FAST as 4 independent processors requesting instructions from the local coprocessor FPGA. On reset or processor power-up, the R3000 starts executing instructions from uncached kernel address space [27]. In this address space, the R3000 communicates using the system bus and can process one instruction every three cycles. This asynchronous interface requires a run, stall, and fix-up cycle for every instruction. With all 4 processors executing independent instruction streams with a CPI of 3, we are able to achieve an aggregate instruction throughput of 33.33 MIPS. Figure 5 shows an oscilloscope trace of the instruction read and execute phase. The top waveform is the 25 MHz system clock generated by the R3000 that is fed back to the FPGA.



```
              X              Y
A——(3) =   721.93 ns     -300 mV
B---(3) =  841.93 ns      3.03 V
      Δ =  120.00 ns      3.33 V
   1/ΔX =  8.3333 MHz
```

Figure 5. Oscilloscope waveform showing 3 instruction fetch cycles.

The middle waveform is the active-low instruction request signal produced by the R3000 and the bottom waveform is the active-low FPGA-generated instruction ready signal indicating to the R3000 that an instruction is on the data bus.

Currently, we are working on the cache interface between the R3000 and the LMC FPGA. We have instantiated a 1K entry instruction and a 1K entry data caches using the FPGA's BRAMs and tested hand coded example programs that exercise loads, stores, and simple integer arithmetic. We use the coprocessor FPGA to issue instructions that cause the processor to jump into cached kernel space. Once in cached kernel space, the LMC FPGA services instruction and data requests from the processor. We have also been able to run trivial programs compiled with gcc or the MIPS SDE toolchain. This method allows us to achieve 100 MIPS on FAST using the preloaded programs ranging from 8 to 53 instructions. Meeting timing constraints has been the major source of errors with respect to FAST's initial processor tile functionality.

The FAST PCB uses 4 different power planes as shown in Table 1. All programmable I/O use 3.3V, while we also supply 5 V directly to the PCB and use DC-to-DC step down voltage regulators to deliver 2.5 V and 1.5 V to the Xilinx FPGA cores. In our initial tests, we consume 3 W of I/O power, 2.5 W of core FPGA, clock distribution and voltage regulator power, and each processor consumes 2.75 W for an overall power dissipation of 16.5 W without FPUs.

4.1. Software Toolchain

FAST uses MIPS processors. The initial programs that were executed in small cache structures on FAST were all hand coded. We have been also able to compile trivial test programs, preload the FPGA caches and execute programs on the processor. We use an SGI version of *gcc* and related SGI software tools or MIPS SDE 6.02, an embedded MIPS software toolchain, to generate statically linked binaries targeting the R3000. Dynamic linking is possible, but requires much more board-level support software. We set the starting text and data segment addresses so that they efficiently map in the cache. We then disassemble the binaries and modify this output to generate initialization files for the FPGA BRAMs acting as 1K entry instruction and data caches. We envision a similar process when running these programs from FAST's main memory, although the initialization file format may change. Section 6 outlines what resources are necessary to execute large programs using this development toolchain.

Currently, there is no operating system for FAST. We are building the initial infrastructure required to run applications. These "infrastructure" Verilog modules serve as the building blocks for future emulated systems, as well as the initial skeleton to support future OS and low-level software development. We envision using PMON, a batch OS for MIPS based development boards [31], for FAST.

5. Related Work

FAST bridges the gap between fully flexible FPGA prototyping arrays [25,35] and application-targeted processor blades and processor subsystems [20,30]. We have built FAST to provide a prototyping substrate for future systems, instead of just exploring design trade-offs for current systems. The synergy exploited by combining the optimized characteristics of FPGAs and microprocessors is also similar to reconfigurable research using configurable components as coprocessors [9]. FAST augments this functionality by manipulating the memory hierarchy as well. Within a processor tile, FAST can be configured to use a variety of cache configurations or other memory structures like FIFOs or stacks. The same is true for the secondary memory, which can also be shared by all the processor tiles or partitioned for processor-tile private memory. While this flexibility exists for FPGA-only systems, the performance of FPGA-only instantiated designs is limited by resource hungry memories, which result in poor resource utilization and slow down performance greatly.

RPM was a prototyping system designed to emulate existing multiprocessors and their various memory design trade-offs [2,6]. RPM, like FAST, is very similar to a cycle-by-cycle RTL simulator. Similarly, both systems use FPGAs as memory controllers. FAST adds to RPM by allowing FPGAs to augment individual processors' core capabilities. Also, FAST targets emulating systems that do not exist, such as HYDRA [8] and Smart Memories [18], facilitating future hardware and software exploration and development. MPOC is another hardware emulation system that used multiple processor cards with a VME bus interconnect [20]. This system uses a MIPS-processor based hardware emulator with architecture reconfiguration abilities limited to signals visible on the VME bus.

Finally, both Altera and Xilinx provide FPGA development boards that feature embedded hard or soft processor cores with a collection of software development tools [28,39]. All the systems that feature FPGAs with hard cores operate with a clock frequency of several hundred megahertz. These systems have several drawbacks that FAST addresses. All the cores lack floating-point units, requiring that floating-point intensive applications use very slow emulation code. All of the cores have primary caches, but there is no visibility within the core to allow modification of these caches. To monitor cache activity, users would have to disable each processor core's cache and instantiate the caches with FPGA resources and buses.

Implementing an external primary cache for embedded hard cores may result in one of two performance penalties. The first option, if possible, is to slow the embedded processor's clock speed down to the speed of the external cache, making the embedded hardcore as slow or slower than FAST's MIPS processors. In this case, the main speed restriction can often be caused by the processor bus interfaces instead of the additional cache and related logic. Alternatively, one can operate the embedded processor at speed, but require the processor to stall on any cache access, including *cache hits*. At the very least, the processor must be stalled, potentially draining the pipeline, and restarted once the cache request can be filled. As a result, the effective "clock rate" will be very slow and hard to control precisely. This second option would require error prone complex accounting and statistics gathering and would simulate unrealistic pipeline and instruction interaction behavior.

Furthermore, the Altera and Xilinx boards do not have FPGAs with 4 embedded processor cores. Thus, if we were using these development options, we would not be able to emulate our initial target architecture [8] with a single development board. The first option outlined above–slowing down the processor to match the external cache interface–yields valid simulation results (if it is possible to slow the embedded processor core), but we believe the performance would degrade further when interfacing multiple development boards to create a 4-processor emulation system. The second option–stalling on every cache access–exacerbates the simulation accuracy problems when interfacing multiple boards. Regardless of the cache implementation options, the FPGA also restricts the size of the cache due to the resource limitations, thereby restricting the architecture exploration. This is not the case with the FAST emulation platform, because it interfaces to actual SRAM chips. On the other hand, the main advantage of these FPGA development boards is the software toolchain and support.

Software simulators have many advantages compared to hardware emulation. Hardware emulation has real capital costs of building the hardware, which is beyond the normal costs required to develop a software simulator. In most cases, simulators are distributed in academia for free, which leads to their widespread use. Even after acquiring *all* donated components, FAST still costs about $5000 to be manufactured and assembled for quantities on the order of tens of boards. Hardware emulation requires additional effort to guarantee that the components, especially FPGAs, meet the timing constraints of the other components. FAST uses 25 MHz MIPS R3000's. These processors have particular timing requirements related to the caches and other processor events. These requirements can be ignored in software simulation, but may reduce the fidelity of the software simulator. If timing is violated, the processors don't execute instructions or the cache access fails for the SRAMs. Software simulators can also be designed to leverage existing software infrastructure, like Simics [17]. In our case, we must develop all the tools from scratch or modify existing software and hardware tools. Finally, the main benefits of hardware emulation are the undeniable simulation fidelity, fast simulation, and application development capabilities.

6. Future Work and Conclusions

We have built FAST to be a flexible, simple, and effective way to emulate chip multiprocessors or similar tightly-coupled multithreaded architectures, even those incorporating features, such as thread-level speculation (TLS), that are not available on any current multithreaded architectures. With the FAST substrate, we can perform conventional validation of these architectures, evaluate *full* benchmark applications that cannot be feasibly simulated, and explore software techniques for dividing programs into threads, all within a reasonable amount of time. Our initial results indicate that FAST can rapidly emulate chip multiprocessor systems. With the processor initialization module complete, we will continue to build Verilog interface modules that can be used as simulation building blocks for multiple configurations.

While the basic design of FAST can handle all of these goals, there are several improvements that could reduce simulation time even further. We are using an embedded microcontroller with an Ethernet port to provide an external FAST communication interface that is very simple but unfortunately often slow. We would like to find an equally simple but faster interface in the future, possibly from existing products like the modules provided by Altera [28]. We would also like to add another level of memory to expand FAST's capabilities. By using DRAM and an associated DRAM controller, we could store more complicated OS and benchmark data on the board, reducing the need for off-board communication — and latency. Another alternative is to addu a daughter card with a Compact Flash module, similar to those modules for Altera development boards [28], directly to the expansion header. This would enable FAST to execute benchmarks that have large memory footprints. We are also currently developing more system software infrastructure to reduce the effort of mapping new architectures onto FAST.

Acknowledgements

We would like to thank Alan Swithenbank and Wade Gupta for their contribution to the FAST PCB layout and fabrication. We would also like to thank Lance Hammond for his comments and initial review of the FAST architecture. This project also received generous donations from Xilinx. This research was supported by National Science Foundation grant CCR-0220138.

References

[1] T. Austin, E. Larson, and D. Ernst "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer Magazine*, Feb. 2002, Page(s): 59 -67

[2] L. A. Barroso, S. Iman, et al., "RPM: a rapid prototyping engine for multiprocessor systems," IEEE *Computer Magazine*, Feb. 1995, Page(s): 26 -34

[3] D. C. Bossen, J. M. Tendler, and K. Reick, "Power4 system design for high reliability," *IEEE MICRO Magazine*, March-April 2002, Page(s): 16 –24

[4] D. Culler and J. P. Singh, *Parallel Computer Architecure A Hardware/Software Approach*, Morgan Kaufmann Publishers, Inc. San Francisco, CA, 1999

[5] J. D. Davis, J. Laudon, K. Olukotun, "Maximizing CMT Througput with Mediocre Cores," *In Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2005

[6] M. Dubois, J. Jeong Y. H. Song, A. Moga, "Rapid hardware prototyping on RPM-2," *IEEE Design & Test of Computers*, July-Sept. 1998, Page(s): 112-118

[7] J. Emer, et. al., "Asim: A Performance Model Framework," *IEEE Computer Magazine*, Feb. 2002, pp: 68-76

[8] L. Hammond, B. Hubbert, et al., "The Stanford Hydra CMP," *IEEE MICRO Magazine*, March-April 2000.

[9] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 24-33, 1997

[10] C. J. Hughes, et al., "Rsim Simulating Shared-Memory Multiprocessors with ILP Processors," *IEEE Computer Magazine*, Feb. 2002, pp. 40-49

[11] J. Huh, et al., "Exploring the Design Space of Future CMPs," *PACT*, pp. 199-210, Sept. 2001.

[12] R. Kalla, B. Sinharoy, J. Tendler, "Simultaneouos Multi-threading Implementation in POWER5," *Hot Chips 15*, Aug 2003

[13] S. Kapil, "Gemini: A Power-efficient Chip Multi-Threaded (CMT) UltraSPARC® Processor," *Hot Chips 15*, Aug 2003

[14] D. Koufaty, D. Marr, "Hyperthreading technology in the netburst microarchitecture," *Micro, IEEE* , Volume: 23 Issue: 2 , March-April 2003, Page(s): 56-65

[15] V. Krishnan and J. Torrellas, "A Chip Multiprocessor Architecture with Speculative Multithreading," *IEEE Transactions on Computers, Special Issue on Multithreaded Architecture*, September 1999

[16] R. Kumar, D. Tullsen, et al., "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance," *The 31st International Symposium on Computer Architecture (ISCA-31)*, June 2004.

[17] P. Magnusson, M. Christensson, J. Eskilson, et al., "Simics: A Full System Simulation Platform," *IEEE Computer Magazine*, February 2002, pages 50-58

[18] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. Dally, and M. Horowitz, "Smart Memories: A Modular Reconfigurable Architecture," *ISCA-27*, June 2000.

[19] D. Matzke, "Will Physical Scalibility Sabotage Performance Gain?," *IEEE Computer Magazine*, September 1997, page(s) 84-88

[20] S. Richardson, "MPOC: A Chip Multiprocessor for Embedded Systems," HPL Technical Report, 2002, http://www.hpl.hp.com/techreports/2002/HPL-2002-186.pdf

[21] M. Rosenblum, E. Bugnion, et al., "Using the SimOS Machine Simulator to Study Complex Computer Systems,"*ACM Transactions on Modeling and Computer Simulations*, vol. 7, no. 1, pages 78-103, Jan. 1997

[22] K. Sankaralingam, R. Nagarajan, et al., "Exploiting ILP, TLP, and DLP Using Polymorphism in the TRIPS Architecture," *ISCA-30*, pp. 422-433, June 2003

[23] G. Sohi, S. Breach, and T. Vijaykumar, "Multiscalar processors," *ISCA-22*, pp. 414–425, June 1995.

[24] J. Steffan and T. Mowry, "The Potential for Using Thread-Level Data Speculation to Facilitate Automatic Parallelization**,"** *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture (HPCA-4)*, February 2-4, 1998.

[25] E. Waingold, et. al., "Baring It All to Software: Raw Machines." *IEEE Compter*, 30(8), pages80-93, September 1997, *MIT/LCS Technical Report TR-709*, March 1997.

[26] D. W. Wall, "Limits of Instruction-Level Parallelism," WRL Research Report 93/6, Digital Western Research Laboratory, Palo Alto, CA, 1993

[27] R3000/R3001 Designer's Guide, Integrated Device Technology, Inc., 1990

[28] Altera Development Kits, http://www.altera.com/products/devkits/kit-dev_platforms.jsp

[29] 128 Megabit (16 M x 8-Bit) CMOS 3.0 Volt-only Uniform Sector Flash Memory with Versatile I/O Control Data Sheet, http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24961.pdf

[30] Artesyn Technologies' Processor Blades and Processor subsystems, http://www.artesyncp.com/products/index.html

[31] PMON5, http://www.carmel.com/pmon/index.html

[32] GNU GDB, http://www.gnu.org/directory/GNU/gdb.html,

[33] 1Mb X 36 S/DCD Sync Burst SRAMs, http://www.gsitechnology.com/8324183672.pdf

[34] High-Speed 3.3V 64K x 36 Asynchronous Dual-Port SRAM,http://www.idt.com/products/pages/Multi-Ports-70V658.html

[35] Mentor Emulation Products, http://www.mentor.com/emulation

[36] RCM3200 RabbitCore User's Manual, http://www.rabbitsemiconductor.com/products/rcm3200/docs.shtml

[37] Fujitsu, Motorola, STMicroelectronics, Synopsys, CoWare and Cadence, *System-C Version 2.0 User Guide,* 2002. Available at http://www.systemc.org

[38] Xtensa Product Brief, http://www.tensilica.com/Xtensa_PB_1003.pdf

[39] FPGA Development Boards, http://www.xilinx.com/

[40] Xilinx Datasheets, http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp