A Desktop Computer with a Reconfigurable Pentium®

SHIH-LIEN L. LU
Intel Corporation
PETER YIANNACOURAS
Universty of Toronto (intern at Intel Corp.)
TAEWEON SUH
Georgia Institute of Technology (intern at Intel Corp.)
ROLF KASSA
Intel Corporation
and
MICHAEL KONOW
Intel Corporation

Advancements in reconfigurable technologies, specifically FPGAs, have yielded faster, more power-efficient reconfigurable devices with enormous capacities. In our work, we provide testament to the impressive capacity of recent FPGAs by hosting a complete Pentium® in a single FPGA chip. In addition we demonstrate how FPGAs can be used for microprocessor design space exploration while overcoming the tension between simulation speed, model accuracy, and model completeness found in traditional software simulator environments. Specifically, we perform preliminary experimentation/prototyping with an original Socket 7 based desktop processor system with typical hardware peripherals running modern operating systems such as Fedora Core 4 and Windows XP; however we have inserted a Xilinx Virtex-4 in place of the processor that should sit in the motherboard and have used the Virtex-4 to host a complete version of the Pentium® microprocessor (which consumes less than half its resources). We can therefore apply architectural changes to the processor and evaluate their effects on the complete desktop system. We use this FPGA-based emulation system to conduct preliminary architectural experiments including growing the branch target buffer and the level 1 caches. In addition, we experimented with interfacing hardware accelerators such as DES and AES engines which resulted in a 27x speedup.

Categories and Subject Descriptors: C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures*

General Terms: Measurement, Performance, Design

Additional Key Words and Phrases: Pentium®, processor, emulator, FPGA, accelerator, simulator, architecture, exploration, model, reconfigurable, operating system

1. INTRODUCTION

Research in computer architecture has traditionally used software simulation of a uni-processor executing a single binary, as in SimpleScalar [Burger et al. 1996]. While improvements to processor pipelines and memory hierarchies were historically very fruitful in this context, more recent demands for increased efficiency, especially in multi-processor environments, requires optimization across the entire system stack (processor architecture, instruction-set, device drivers, operating system, and applications). However system-level research is stifled by the slow simulation speeds and/or lack of detailed modelling inherent in the software simulators traditionally used to innovate in microprocessor systems.

Field Programmable Gate Arrays (FPGAs) are seen as the solution to this problem and are being targetted in the development of a new research infrastructure which not only simulates a complete system, but a multi-processor one [Gibeling et al. 2006]. The flexibility, speed (of both development time and simulation time), and enormous capacity of FPGAs qualifies them for the emulation of microprocessor systems. However there are two major obstacles to using FPGAs as emulation systems: (i) the lack of support for real and modern-day operating system software; (ii) the expensive cost of custom multi-FPGA boards necessary to host the large/complex microprocessors; Our work provides compelling evidence that both can be overcome.

We emulate a version of a commercial x86 desktop processor on an FPGA and run real operating systems on stock hardware [Lu et al. 2007]. To be precise, we've replaced a Pentium®[1] microprocessor from its standard socket on a stock motherboard, with a single Xilinx Virtex-4 LX200 FPGA which implements the Pentium® core. The stock motherboard with a standard socket is underclocked at 25 MHz and all system components such as memory, graphics card, CDROM, hard disk, mouse and keyboard can be operated at the same relative speeds as in an original system. Most importantly, our Pentium® emulation system provides us the ability to run real operating systems, such as Fedora Core 4, Red Hat 9, and Windows XP on the FPGA while interacting with real hardware components.

Our system provides an interesting proof-of-concept demonstrating that microarchitecture research can be done with full detail and completeness at the speeds of FPGA-implemented hardware, and in addition, can be performed with the realism of commercial processors running modern-day operating systems while remaining relatively inexpensive. The issue of support for modern operating systems including existing closed-source binaries such as Windows has researchers looking at binary translation as a solution [Gibeling and Wawrzynek 2006]. We demonstrate that acquiring this support is not only feasible, but needs only a single FPGA device, some commodity hardware, and a small and relatively simple board mostly for translating the FPGA pinout to Socket 7 compatibility.

The ability to host desktop microprocessors on an FPGA device and have it execute modern-day consumer softwares has significant ramifications for the reconfigurable systems community. It may not be feasible for desktop processors to be hosted on FPGAs commercially, but with academia and industry embracing the concept as a research vehicle, at the very least, researchers will discover innovative ways to use the reconfigurable fabric (for example by adding custom instructions or parameterizing parts of the architecture), which may then pave the way for reconfigurable technologies to be more tightly integrated into mainstream processor devices. For that reason, researchers and vendors in reconfigurable technology are well-motivated to encourage processor exploration on reconfigurable devices, and

[1]Pentium® is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

our work provides compelling evidence of its fruitfulness.

The FPGA-based Pentium® desktop system provides a powerful tool for the exploration and customization of future microprocessors. Although the system being emulated does not contain a state-of-the art microprocessor, its applicability to modern architectural research has recently spiked due to the successful arrival of chip multi-processors (CMPs). As the number of cores in a CMP increases, the design of the interconnect between the cores rivals in importance the architecture of the individual cores. The only requirement often being that the cores can support real application software. Our emulation system has already been expanded to a multiprocessor system by using available dual socket motherboards, though that work is still in progress. In addition, further effort in improving the mapping of the processor to FPGA resources, coupled with the substantial growth in capacity with newer FPGA families such as Virtex-5, will lead to relevant multiprocessor exploration systems on even a single device.

In this work we make the following contributions: (i) we demonstrate the enormous capacity of modern FPGAs by hosting a complete Pentium® processor on a single Virtex-4 FPGA and executing modern operating systems on stock hardware (ii) we analyze the Pentium® core implementation on the FPGA and crudely contrast it to its implementation using the silicon technology of its time, (iii) we demonstrate the emulator's ability to measure the effect of microarchitectural changes on the complete system using the SPEC2000 integer benchmarks by performing some preliminary architectural enhancements—specifically we parameterize the branch target buffer and the L1 cache; and (iv) we demonstrate its ability to be customized by experimenting with adding hardware accelerators such as AES and DES crypto engines.

The remaining sections of this document will summarize related work and relevant background in Section 2, describe the Pentium® emulation system in more detail in Section 3, outline the implementation of our architectural enhancements made in Section 4, discuss the area/speed effects of the architectural implementations in Section 5, and then conclude in Section 6.

2. BACKGROUND

Microprocessor research is typically done through instruction set simulators such as SimpleScalar [Burger et al. 1996], but detailed simulation can take up to a month [Wunderlich et al. 2004] for a single SPECINT2000 benchmark. Researchers are forced to trade fidelity of simulation results by reducing the model's detail/completeness for timeliness. Our FPGA-based Pentium® can execute all 12 SPECINT2000 benchmarks in just over 3 days, and emulates the complete processor design at the cycle accurate level necessitated by hardware design.

The Bochs open-source IA-32 emulator simulates behaviorally at an instruction level and can achieve rates of 21-25 million instructions per second on a 2.6GHz Core2 Duo [Butler 2006]. Our system running at 25 MHz can achieve 50 million instructions per second for the dual-pipeline Pentium®, a speed that is twice as fast as Bochs but also captures all hardware details in the complete processor design. These and other full-system software simulators such as Simics [Magnusson 2002] and SimOS [Rosenblum et al. 1995] can provide impressive simulation speeds with simple abstract behavioral models, but are more suited for studying software applications rather than researching architecture. Successful innovation in computer architecture must also complement many decades of previous enhancements, requiring detailed (cycle-accurate) modelling of all processor components and the interactions between them. This level of detail significantly slows simulation speeds as more things need to be modelled causing more sequential code to be written and executed. In contrast, our system can easily scale to more components or even processors, with only occasional reductions in clock frequency and limited only by the size of the FPGA (which is increasing with Moore's Law).

The concept of using FPGAs to more quickly and more accurately explore the microprocessor design space has recently gained traction causing publications on the topic to multiply [WARFP 2005]. Some of this work focusses on accelerating simulation times by offloading highly detailed resource modelling into the FPGA while a software simulator remains the core of the emulation environment [Chiou et al. 2006; Wunderlich and Hoe 2004]. Other research often focusses on a single architectural novelty (for example transactional parallel systems [Kozyrakis and Olukotun 2005], caching [Lu et al. 2005], vector-thread processors [Kasper et al. 2005]) and build FPGA-based models of the relevant hardware. Contrary to both these approaches, we implement the complete microprocessor on an FPGA making the entire processor architecture flexible.

Complete RTL models of microprocessors have already become available for the SPARC V8 [Gaisler 2003], Niagara [Sun Microsystems 2006]. These cores can be synthesized to FPGAs and are designed to facilitate design space exploration [Jones et al. 2004]. However, to the best of our knowledge, we are the first to employ such a core in a real desktop system with real hardware peripherals capable of hosting real and modern operating systems. FPGA implementations of microprocessors have been worked on for more than a decade[Gray 2000], however these processors are often simple cores meant more for exploring FPGA design issues than processor architecture. Our work emphasizes realistic architecture exploration and hence plugs into a real hardware system and supports the advanced features of modern day operating systems (such as virtual memory).

An abundance of research already exists in the embedded domain which applies customization to an FPGA-based core. The fruitfulness of application-specific microarchitectural variation was commercialized by Tensilica [Tensilica 1997] and can be seen in [Yiannacouras et al. 2006] and its automatic navigation in [Sheldon et al. 2006]. In addition, the effect of including custom instructions into such cores was explored [Biswas et al. 2006]. While our work is similar in spirit to these works we differentiate ourselves by focussing on the desktop domain and emphasizing peripheral and operating system interaction.

Commercial products such as Cadence's Incisive Palladium [Cadence 1988] allow in-circuit emulation of arbitrary RTL circuits. These systems are usually big boxes with multiple boards and multiple FPGAs per board. They are very expensive and require additional wiring to hardware that the circuit needs to interface with. Our system is low cost and is already designed to plug into a Socket 7 motherboard.

A complete Commodore 64 has also been ported to FPGA and can run in a Commodore box with its own operating system[Ellsworth 2007]. However while

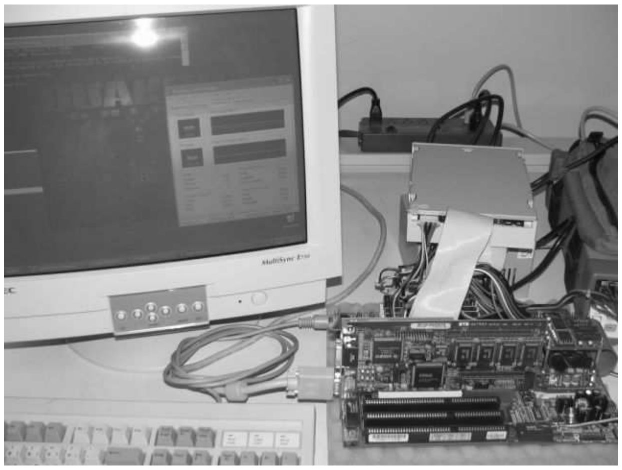

Fig. 1. Image of the FPGA-based processor emulator system equipped with standard hardware peripherals, a Xilinx Virtex-4 device in place of a microprocessor chip, all running Windows XP

similar in concept, the 16-bit processor is too old to be relevant for state-of-the-art computer architecture research.

3. THE FPGA-BASED PENTIUM® EMULATION SYSTEM

The complete emulation environment consists of four main components: (i) the FPGA which hosts the Pentium® processor; (ii) the hardware including motherboard and peripherals; (iii) the software/operating system; and (iv) the necessary FPGA CAD software required to implement the FPGA design. We discuss each of these four items in further detail.

3.1 The Processor

The RTL which describes the processor is written in VHDL in a structural form. Some manual mapping to FPGA memories was performed and many of those components are easily parameterized, but much of the existing HDL is not. Modification to the processor requires some studying and visualization of the affected circuitry. This tends to encourage more isolated changes to the datapath, but more sophisticated programmability is available using microcode sequences in place of modifying instruction decoding logic. Improvements to the VHDL code to make it more amenable to modification is still ongoing.

The processor implemented in our emulation system is the original Pentium® which is the desktop processor released after the 486 and before the Pentium Pro®. The 3.3 million transistor processor was released in 1994 in a 0.6 micron technology and was originally clocked at 75 MHz [Intel Corporation 1997]. It is a 32-bit in-order 5-stage dual-pipeline processor supporting the IA32 instruction set including floating point instructions using an on-chip pipelined floating-point module. It is equipped with two on-chip separate 8 KB 2-way set associative level 1 caches for data and instructions and implements the MESI protocol for use in multiprocessor environments. It also includes dynamic branch prediction using a

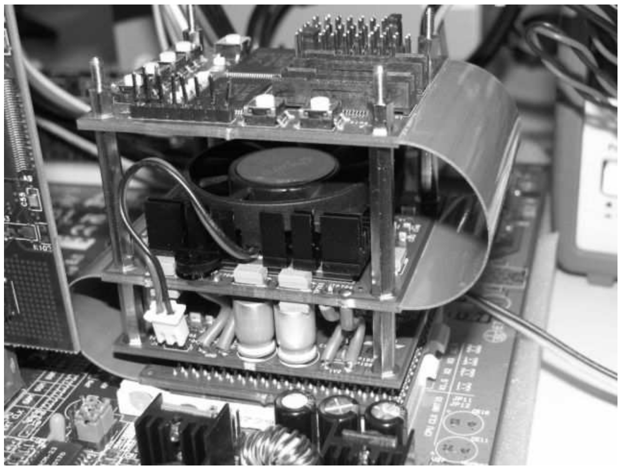

Fig. 2. Image of the 3-level stacked board which houses the Xilinx Virtex-4 and converts it for use on the processor motherboard.

256 entry predictor table and branch target buffer.

A 3-level stacked board houses the FPGA and necessary circuitry. The first level contains the pin/power conversion between the motherboard and FPGA allowing it to be plugged directly into the motherboard. The second level contains the FPGA itself, and the top level contains the programming circuitry for the FPGA. The FGPA used to host the Pentium® is a Xilinx Virtex-4 LX200 90 nanometer device, which gets less than half consumed by the Pentium®. More detailed analysis of the Virtex-4 resources utilized by the Pentium® will follow in Section 5.

3.2 The Computer Hardware

Everything other than the actual Pentium® chip is original hardware that would typically be used in a Pentium system. The motherboard is an original ASUS Socket7 motherboard with 196 MB of SDRAM, and original chipset and BIOS. The only modification is that the board's clock is underclocked to approximately 25 MHz—one third of the speed the system was designed for. Note that the underclocking of the board affects the processor, RAM, cache, chipset, and bus speeds and hence preserves the relative speeds of the original system. Other peripherals attached to the board include graphics card, USB connector, hard disk, CDROM, keyboard, mouse, and monitor.

3.3 The Operating Systems

The most powerful ability of our FPGA-based system is its ability to boot real operating systems. We successfully installed unmodified versions of Fedora Core 4, Red Hat 9, and Windows XP on the Pentium®; the installation procedure was no different than on any typical desktop system. In terms of performance and usability, it takes approximately 10 minutes to boot Fedora Core 4 without a GUI. Command shells, and text editors such as vim operate just as expected on a modern computer system, and GCC can compile small programs in seconds. Typing is certainly done at full speed, searches through normal sized text files succeed with unnoticeable latency. In summary, the system is perfectly usable as a desktop computer for very simple non-graphical applications.

3.4 FPGA Development

To synthesize the Pentium® we use Synplify Pro 8.5.1 for high-level synthesis of the VHDL and then use Xilinx ISE 8.1i for placement and routing onto the Virtex-4 device. The entire process takes between 10 and 20 hours to synthesize, map, place, route and generate a bitstream, followed by an additional 20 seconds to download the bitstream to the device. This turnaround time is orders of magnitude quicker than the fabrication time for a silicon implementation of the processor which could be inserted directly on the motherboard. In terms of debugging, Modelsim 6.1 is used to simulate the VHDL in lockstep with a software simulator which models the original behavior of the processor. A suite of regression tests are used to ensure the processor is still a functional x86 machine. The regression tests are a subset of those used to verify the original Pentium®.

4. MODIFYING THE PENTIUM® CORE

In this section we discuss our design and implementations of the three different enhancements we made to the Pentium®. Below we discuss the expansions made to the branch prediction capabilities and the L1 cache of the core, and we detail our integration of the hardware acceleration for encryption/decryption through our AES and DES crypto-engine.

4.1 Expanding the Branch Target Buffer

The Pentium® is equipped with dynamic branch prediction which consists not only of a predictor table to speculate on conditional branches, but also a branch target buffer (BTB) to speculate on indirect branches—when the target of a branch can not be deduced by the current program counter and the instruction word alone (ie. the branch target cannot be resolved early enough in the pipeline), the Pentium® uses the BTB to guess where the branch will jump to. The branch target buffer originally held 256 entries allowing it to speculate "correctly" for up to the last 256 indirect branches. The size of the BTB was doubled to 512 entries and no other changes were required to the rest of the system to accommodate this growth.

4.2 Expanding the L1 Caches

The Pentium® 8 KB L1 caches are very small by today's standards. There are two such caches, one for data memory, the other for instruction memory, each of which are 8 KB and 2-way set associative with 32 bytes per cache line. The caches were increased internally by 4x way-wise to become 32 KB 8-way set associative caches. The LRU replacement policy which determines which line gets evicted within a full set was also expanded to handle the sets of 8 cache lines. Both instruction and data caches can be individually configured to either the 8KB or 32KB versions, but in this work we always keep them the same size.

4.3 Integrating AES and DES Crypto Engine

We integrated two crypto-engines into the Pentium®: Advanced encryption standard (AES) and data encryption standard (DES). Security has more recently become a critical requirement in many computing areas such as network security and digital rights management. To support such security requirements and maximize system performance, security-enhanced processors are preferred and becoming available in the market [Hifn Inc. 2006]. In our approach we integrate custom instructions for accelerating encryption and decryption directly into the processor.

We retrieved AES and DES intellectual property (IP) cores from Opencores [Opencores 2007]. The AES core implemented the Rijndael's algorithm and takes a 128-bit key and a 128-bit plaintext/cyphertext for encryption and decryption, respectively. The DES core takes a 56-bit key and 64-bit plaintext/cyphertext for encryption and decryption, respectively. In our implementation, we extended the x86 ISA to integrate AES and DES engines by creating new Model-Specific Registers (MSRs)—a set of hidden registers usually used to capture debug/performance information which are accessible only by two privileged instructions called `rdmsr` and `wrmsr` respectively for reading and writing. We can use the MSRs to provide communication with the crypto-engines. That is, the encryption/decryption is executed by sending data to the appropriate crypto-engine by “writing” to our newly created MSR(s) via the `wrmsr` instruction, then the corresponding cyphertext or plaintext result can be “read” from the crypto-engine via the `rdmsr` instruction. Similarly, control information is sent to the crypto-engines using another MSR. For example, users can choose the configuration such as AES or DES, encryption or decryption, and key or input data. This approach reduces the access latency by avoiding comparably expensive bus accesses had the engine been a co-processor connect through the bus.

Implementing the new MSRs involved several changes. First the actual MSRs and necessary logic to access them was inserted into the VHDL design. Second the privilege protections checks were removed from `rdmsr` and `wrmsr` allowing us to access the crypto-engines from user space rather than through the operating system. Finally, many optimizations were required to improve the execution speed of these instructions since generally `rdmsr` and `wrmsr` are very slow instructions. With all these modifications we achieved a communication overhead of only 6 cycles between the processor and the crypto-engines (the engines were clocked at the same CPU frequency though capable of much higher clock rates). The entire design time was less than two weeks for this change and involved modifications to the microcode in addition to VHDL changes to only one isolated component.

5. EXPERIMENTING WITH THE PENTIUM® SYSTEM

In this section we analyze and benchmark the FPGA-based Pentium® system to extract the following results: (i) an area breakdown of the Pentium® as reported by the CAD flow; (ii) a comparison between the original branch target buffer and our expanded version; (iii) a comparison between the original 8KB L1 cache and our expanded 32KB L1 cache; (iv) an analysis of the crypto-engine hardware accelerator. We examine each of these in more detail. Note that we report on

Table I. Virtex-4 resource utilization by the unmodified Pentium®.

Resource	Number used	Percent Used
4-LUTs	65615	37%
Registers	26859	15%
Slices	41438	46%
DSP48s	29	30%
BRAMs	118	35%

area in terms of Virtex-4 resources but are cognizant that these results may not predictably map to a real silicon implementation. Nonetheless the area analysis can be used for first-order approximations.

5.1 Area Breakdown of the Pentium®

We synthesized the Pentium® VHDL to the Virtex-4 LX200 and noticed that less than half of the device resources were used; the corresponding data is shown in Table I taken after high-level synthesis and technology mapping was completed. Only 37% of the LUTs were used to store all the logic for the Pentium®, however they were distributed through 46% of the slices. Also, 35% of the block RAMs were utilized (distributed RAMs are counted as 4-LUTs). With more than half of the resources still available, there exists sufficient space on the device for expanding and augmenting the Pentium®.

Figure 3 shows the breakdown of each Virtex-4 resource used by different units in the processor; the data was collected from the synthesis results reported by Synplify Pro. All of the DSP48 (multipliers) were used by the floating point unit, and nearly all of the block RAMs were divided amongst the instruction cache, data cache, and microcode units. The Virtex-4 LUTs were used mostly by the FPU, ALU, address generation, and caches. The entire memory hierarchy (including the caches and bus interface) claimed approximately 45% of the LUTs used, suggesting that even when considering only logic, almost half of the chip is devoted to communication leaving the other half for control and actual computation.

An area constrained architect can presumably use off-chip memory to emulate the ”on-chip caches” which would free nearly half the FPGA resources used by the processor. This approach can be valid since the processor implemented on an FPGA would likely be clocked considerably slower than memory devices, allowing the multi-cycle memory latency to occur within a single (longer) processor cycle. Many low cost FPGA boards have on-board SRAM/SDRAM/DDR-RAM suitable for this task, so architects/researchers can use these low-cost boards for their exploration rather than purchasing expensive custom boards or multi-FPGA boards.

Although synthesizable, the Pentium® VHDL was not designed for mapping to an FPGA. Recent work [Gibeling and Wawrzynek 2006] suggested that a processor designed specifically for synthesis to an FPGA can be more than an order of magnitude smaller than a generically written mostly-behavioral VHDL processor. While our processor has had some manual tweaking to guide its mapping to some FPGA resources, we too also believe that the resource usage of the Pentium® can be significantly reduced by more carefully mapping structures to the resources in the FPGA. Of particular note is the mapping to block RAMs. The interconnection

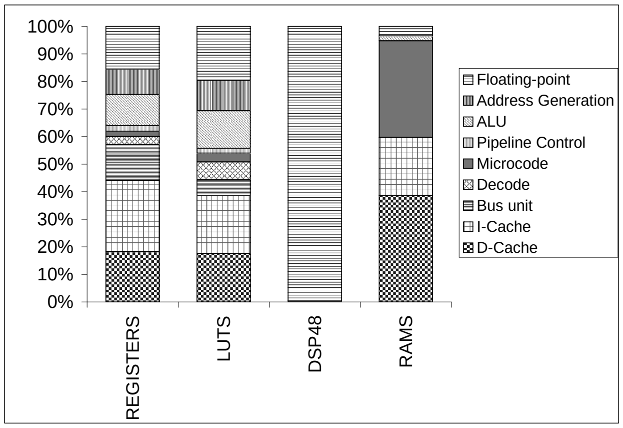

Fig. 3. Breakdown of FPGA resources used by different parts of the Pentium® archtiecture.

between large numbers of under-utilized BRAMS is a major contributor to both the speed and area overhead. Multiple BRAMs are often required due to limitations on the number of ports or the width of the ports. Re-architecting the processor to better utilize the block RAMs may be of great benefit to the FPGA design.

In spite of the core's ill-suitedness for FPGA design, it still provides an interesting point of comparison for FPGAs as a platform. Recent work [Kuon and Rose 2006] has measured FPGAs to be 3x slower in speed and 35x larger in area compared to a standard cell ASIC flow with both using 90nm technology. With some simple and crude calculations we can attempt to do the same with the 12 year old Pentium®, although it was highly hand-optimized while the previously cited work used a push-button design flow. The FPGA-based core is clocked at 25 MHz compared to the 75 MHz it originally ran at 12 years ago, meaning the 90nm FPGA is already 3x slower than the older 600nm silicon technology. Accounting for the generation gap can only be crudely estimated so we do not do so here. Nonetheless, we see that compared to a highly optimized transistor design, the push-button FPGA flow not only eliminates 12 years of electrical innovation but is an additional 3x slower. We expect more careful mapping to FPGA resources may reduce this penalty.

5.1.1 *Future FPGA Architectures.* Future FPGA devices will rapidly grow in capacity. The Virtex-4 LX200 used in our research was the largest FPGA in that family with 200,000 logic cells. The new Virtex-5 family has capacities of up to 330,000 logic cells. Such a device can almost host four of our Pentium® designs which would make it a great research platform for recently emerged quad core microprocessors. Additional fine-tuning of the VHDL design to aid its mapping to FPGA technology would make that a reality and may also yield even higher processor densities in FPGA devices.

In addition to the size, the architecture of an FPGA also plays a large role in the ability to map a processor design to a given FPGA device and hence is a critical component to an FPGA-based processor emulation system. We hope that future FPGA architectures will better accommodate processor designs. The Virtex-5 architecture may boost logic density because of the new 6-input LUT size, but

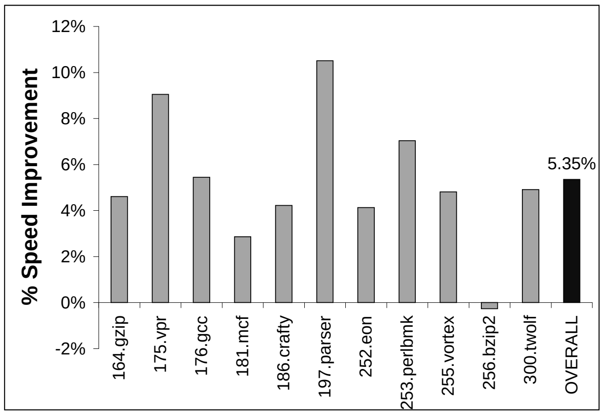

Fig. 4. Performance increase of the doubled branch target buffer on SPEC2000 integer benchmarks.

new hard circuits could be very useful to processor designs. Block RAMs used for caches, register files, memory management units, etc. are typically constrained to 2 ports while more would prevent the manual intervention required to stop the CAD tool from mapping those components to structures built expensively out of flip flops. A crossbar circuit could prove interesting for multiprocessor systems.

5.2 Comparing Branch Target Buffer Sizes

Doubling the branch target buffer should give the processor twice as many branch target entries, which ideally should translate to increased accuracy in predicting taken indirect jumps. This modification was a simple warm-up exercise requiring only an extra block RAM and a small amount of logic most likely a side-effect of the randomness in the CAD algorithms. The performance of the expanded BTB was measured across all SPEC2000 integer benchmarks. Since the system is in fact real, the time to complete a single benchmark run is non-deterministic and takes almost a day making it difficult to average out the non-determinism. As such, some of the effects of the increased branch target buffer remain hidden in this noise.

Figure 4 shows significant speed improvements up to 11% by PARSER. VPR and PERLBMK also benefit largely from the increased predictor accuracy. On average the expanded BTB provides a 5.35% speed improvement, which is quite significant for such a small change.

5.3 Comparing Level 1 Cache Size

Figure 5 shows the additional FPGA resources consumed from growing the L1 caches from 8KB (2-way) to 32KB (8-way). Almost 25% more logic was necessary for the expansion as well as more than 50% more block RAMs making this growth in L1 cache very expensive with respect to area. In addition to the area cost, the place and route time is more than doubled. Nonetheless the performance benefit is quite substantial.

Figure 6 plots the performance improvement of the expanded L1 cache for each SPEC2000 integer benchmark. An average of 16% performance improvement is

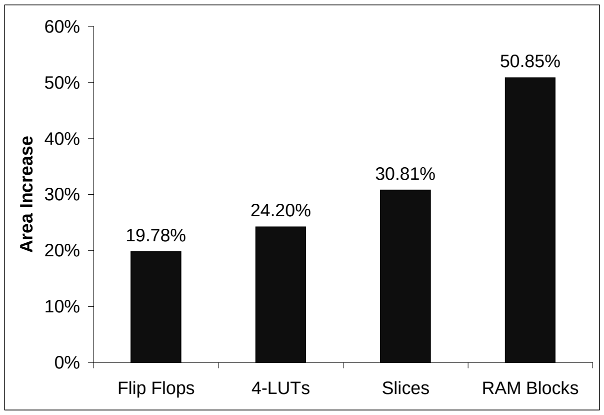

Fig. 5. Area increase of the 32KB 8-way L1 caches versus the 8KB 2-way L1 caches.

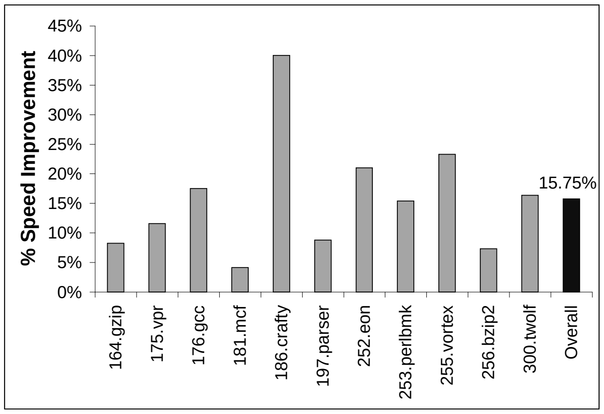

Fig. 6. Performance increase of the 32KB 8-way L1 caches versus the 8KB 2-way L1 caches.

achieved with benchmarks such as CRAFTY reaching as high as 40%. Although there are a myriad of cache studies, we believe this work is unique in capturing operating system effects such as cache flushes and preemption while sustaining high simulation speeds.

5.4 Evaluating the Crypto-Engine

The AES takes only 12 CPU cycles to finish its computation for encryption/decryption, and the DES takes 16 CPU cycles, both significantly faster than a software implementation. The best known software implementation for AES written specifically for the same Pentium® executes in 320 cycles [Granboulan 2000]. This results in an execution speedup of 27x for our custom crypto-engine versus the best software implementation. Table II summarizes the resource utilization of the AES and DES engines on the Virtex-4 FPGA and shows that the logic requirement is very small but a substantial number of BRAMs were required. Nonetheless, for secure environments the performance improvement may justify the extra resources.

Table II. Virtex-4 resource utilization of the AES and DES IP cores.

Resource	Number used
4-LUTs	2347
Registers	1319
DSP48s	0
BRAMs	72

6. CONCLUSION

The FPGA-based Pentium® emulator is a powerful tool for researching desktop processor architectural enhancements. Its ability to quickly prototype architectural changes and measure their effects at the application-level in the presence of a real operating system provides a more realistic research tool without the expensive costs and long design times associated with actually creating a silicon implementation. Such a system can be used to optimize across the entire system stack: architecture, instruction-set device drivers, operating systems, and applications without the prohibitive simulation times of a software simulator.

Although the Pentium® processor requires a large number of FPGA resources, modern FPGA devices have ample capacity for hosting desktop uniprocessors. The Pentium® processor consumed less than half of the Virtex-4 LX200 providing significant space for growth of the design. We expect that more careful mapping of the design to FPGA resources will result in large-scale reductions in size. Moreover, the improvements to capacity and logic architectures of future generation FPGAs will only make them more capable of hosting larger more recent processor designs, including multiprocessors.

The Pentium® emulator was used to explore expansion of both the branch target buffer and L1 cache. Doubling the size of the branch target buffer required a negligibly small amount of resources and provided significant speed improvements on the SPEC2000 integer benchmarks. The L1 caches were quadrupled from 8KB 2-way to 32KB 8-way caches and required significantly more FPGA resources and synthesis time, but also provided large speedups for the SPEC2000 benchmarks.

Finally we integrated custom instructions to interface with AES and DES crypto engines to accelerate encryption and decryption operations. The tight integration of these engines into the core provided instructions which can encrypt/decrypt in 12-16 CPU cycles providing a 27x speedup over the best known software implementation. The added area was relatively small for the engines.

REFERENCES

Biswas, P., Banerjee, S., Dutt, N., Ienne, P., and Pozzi, L. 2006. Performance and Energy Benefits of Instruction Set Extensions in an FPGA Soft Core. In *IEEE International Conference on VLSI Design (VLSID)*. IEEE.

Burger, D., Austin, T. M., and Bennett, S. 1996. Evaluating future microprocessors: The simplescalar tool set. Tech. Rep. CS-TR-1996-1308.

Butler, T. R. 2006. *Bochs.* http://bochs.sourceforge.net/doc/docbook/user/bochsrc.html.

Cadence. 1988. Cadence Incisive Palladium. http://www.cadence.com/quickturn/.

Chiou, D., Sunjeliwala, H., Sunwoo, D., Xu, J., and Patil, N. 2006. FPGA-based Fast, Cycle-Accurate, Full-System Simulators. In *Workshop on Architecture Research using FPGA Platforms in the 12th International Symposium on High-Performance Computer Architecture.*

Ellsworth, J. 2007. C One. http://c64upgra.de/c-one/.

Gaisler, G. 2003. LEON SPARC. http://www.gaisler.com.

Gibeling, G., Schultz, A., and Asanovic, K. 2006. The RAMP Architecture and Description Language. In *Workshop on Architecture Research using FPGA Platforms in the 12th International Symposium on High-Performance Computer Architecture*. Austin, Texas.

Gibeling, G. and Wawrzynek, J. 2006. A Universal Processor for RAMP. Tech. rep., http://ramp.eecs.berkeley.edu/index.php?publications.

Granboulan, L. 2000. AES Timings of the Best Known Implementations. http://www.di.ens.fr/ granboul/recherche/AES/timings.html.

Gray, J. 2000. Designing a Simple FPGA-Optimized RISC CPU and System-on-a-Chip.

Hifn Inc. 2006. *4450 HIPP III Storage Security Processor*. Hifn Inc.

Intel Corporation 1997. *The Pentium Datasheet*. Intel Corporation, http://www.intel.com/support/processors/pentium/.

Jones, P., Padmanabhan, S., Rymarz, D., Maschmeyer, J., Schuehler, D. V., Lockwood, J. W., and Cytron, R. K. 2004. Liquid Architecture. In *International Parallel and Distributed Processing Symposium: Workshop on Next Generation Software*.

Kasper, J., Krashinksy, R., Batten, C., and Asanovic, K. 2005. A Parameterizable FPGA Prototype of a Vector-Thread Processor. In *Workshop on Architecture Research using FPGA Platforms in the 11th International Symposium on High-Performance Computer Architecture*.

Kozyrakis, C. and Olukotun, K. 2005. ATLAS: A Scalable Emulator for Transactional Parallel Systems. In *Workshop on Architecture Research using FPGA Platforms in the 11th International Symposium on High-Performance Computer Architecture*.

Kuon, I. and Rose, J. 2006. Measuring the Gap Between FPGAs and ASICs. In *Proceedings of the 2006 International Symposium on Field-Programmable Gate Arrays*. ACM Press.

Lu, S.-L., Nurvitadhi, E., Hong, J., and Larsen, S. 2005. Memory Subsystem Performance Evaluation with FPGA based Emulators. In *Workshop on Architecture Research using FPGA Platforms in the 11th International Symposium on High-Performance Computer Architecture*.

Lu, S.-L. L., Yiannacouras, P., Kassa, R., Konow, M., and Suh, T. 2007. An FPGA-based Pentium®In A Complete Desktop System. In *FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*. ACM Press, New York, NY, USA, 53–59.

Magnusson, P. S. 2002. Simics: A Full System Simulation Platform. *IEEE Computer 35,* 2, 50–58.

Opencores. 2007. Opencores.org. http://www.opencores.org.

Rosenblum, M., Herrod, S. A., Witchel, E., and Gupta, A. 1995. Complete Computer System Simulation: The SimOS Approach. *IEEE parallel and distributed technology: systems and applications 3,* 4 (Winter), 34–43.

Sheldon, D., Kumar, R., Lysecky, R., Vahid, F., and Tullsen, D. 2006. Application-Specific Customization of Parameterized FPGA Soft-Core Processors. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM Press.

Sun Microsystems 2006. *OpenSPARC*. Sun Microsystems, http://opensparc.sunsource.net/.

Tensilica 1997. *Xtensa*. Tensilica, http://www.tensilica.com.

WARFP. 2005. Workshop on Architecture Research using FPGA Platforms. International Symposium on High-Performance Computer Architecture, San Francisco, California.

Wunderlich, R. and Hoe, J. C. 2004. In-system fpga prototyping of an itanium microarchitecture. In *International Conference on Computer Design (ICCD)*.

Wunderlich, R. E., Wenisch, T. F., Falsafi, B., and Hoe, J. C. 2004. An Evaluation of Stratified Sampling of Microarchitecture Simulations. In *In Proceedings of the Third Annual Workshop on Duplicating, Deconstructing, and Debunking (ISCA-31)*.

Yiannacouras, P., Steffan, J. G., and Rose, J. 2006. Application-Specific Customization of Soft Processor Microarchitecture. In *FPGA '06: Proceedings of the 2006 international symposium on Field-programmable gate arrays*. ACM Press.