An FPGA-Based Pentium® in a Complete Desktop System

Shih-Lien L. Lu
Intel Corp.
shih-lien.l.lu@intel.com

Peter Yiannacouras*
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
*While interning at Intel.
yiannac@eecg.utoronto.ca

Taeweon Suh*
Department of Electrical and Computer Engineering
Georgia Institute of Technology
*While interning at Intel.
suhtw@ece.gatech.edu

Rolf Kassa,
Michael Konow
Intel Corp.
michael.konow@intel.com

ABSTRACT

Software simulation has been the predominant method for architects to evaluate microprocessor research proposals. There are three tenets in modeling new designs with software models: simulation speed, model accuracy and model completeness. The increasing complexity of the processor and accelerated trend to have multiple processors on a chip are putting burden on simulators to achieve all tenets mentioned, including accurately capturing OS effects. In this work we perform preliminary experimentation/prototyping with an emulation system which overcomes the tension to satisfy all three requirements. The system is 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®[1] 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 27x speedups.

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

FPGA'07, February 18–20, 2007, Monterey, California, USA.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures*

General Terms

Measurement, Performance, Design

Keywords

Pentium®, processor, emulator, FPGA, accelerator

1. INTRODUCTION

Research in computer architecture has traditionally used software simulation of a uni-processor executing a single binary, as in SimpleScalar [5]. While improvements to processor pipelines and memory hierarchies were historically very fruitful in this context, more recent demands for increased efficiency requires optimization across the entire system stack (processor architecture, instruction-set, device drivers, operating system, and applications) with multiple processors. However system-level research is stifled by the slow simulation speeds/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 [9]. The flexibility, speed (of both development time and simulation time), and enormous capacity of FPGAs qualifies them for the emulation of microprocessor systems. However, one of the toughest issues facing the development of such an emulation system is compatibility for existing operating systems. FPGA vendors have designed processor cores which are very small and simple, but have limited support for even barebone embedded operating systems. In addition, the desire to run existing OS binaries including closed-source ones such as Windows has researchers looking at binary translation as a solution [10].

In this work we emulate a version of a commercial x86 desktop processor on an FPGA to run real operating systems on stock hardware. To be precise, we've replaced a Pentium® microprocessor from its standard socket on a stock motherboard, with a single Xilinx Vritex4 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, USB devices, mouse and keyboard can be operated at the same relative speeds as in an original system. Most importantly, our FPGA-based 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.

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, system level architectural decisions are becoming more important. Our emulation system has already been expanded to a multiprocessor system by using available dual processor motherboards, though that work is still in progress.

In this work we make the following contributions: (i) we analyze the Pentium® core implementation on the Virtex-4 FPGA and crudely contrast it to its implementation using the silicon technology of its commercial debut, (ii) we perform preliminary architectural enhancements which demonstrate the emulator's ability to measure the effect of microarchitectural changes on the complete system using the SPEC2000 integer benchmarks—specifically we parameterize the branch target buffer and the L1 cache; and (iv) we experimented with adding hardware accelerators such as AES and DES.

The ability to place desktop microprocessors on an FPGA device and have it execute consumer applications has significant ramifications for the FPGA 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 programmable FPGA fabric (for example by adding custom instructions or parameterizing parts of the architecture), which may then pave the way for FPGA fabric to be tightly integrated into ordinary desktop processor devices. Also, it provides an interesting point of comparison allowing us to benchmark modern FPGA technology against twelve year old transistor-based silicon technology.

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

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 [14]. 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 [7]. Other research often focusses on a single architectural novelty (for example transactional parallel systems [17], caching [19], vector-thread processors [16]) 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 [1], Niagara [3], and PowerPC [4]. These cores are can be synthesized to FPGA and are designed to facilitate design space exploration as seen by Jones et al [15]. 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. Our emulation platform also provides several orders of magnitude of simulation time speedup over software emulators such as Simics [8] and SimOS [20].

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 seen in [22] and its automatic navigation in [21]. In addition, the effect of including custom instructions into such cores was explored [6]. 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.

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 processor used 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 [13]. 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 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

Figure 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

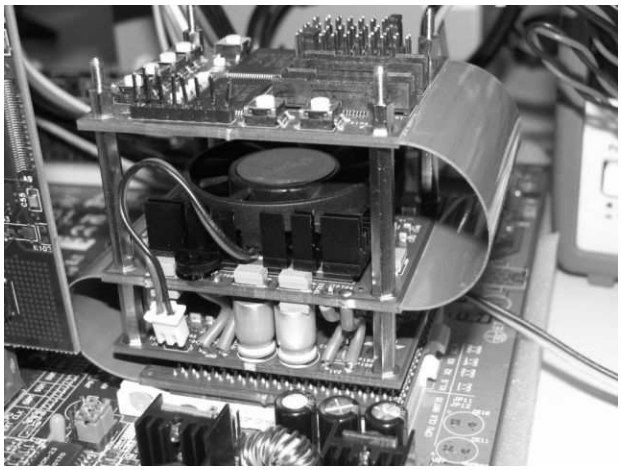

Figure 2: Image of the 3-level stacked board which houses the Xilinx Virtex-4 and converts it for use on the processor motherboard.

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 behaviour 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 [12]. 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 [2]. 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.

Table 1: 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%

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 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 1 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.

Although synthesizable, the Pentium® VHDL was not designed for mapping to an FPGA. Recent work [10] 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-behavioural 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 between large

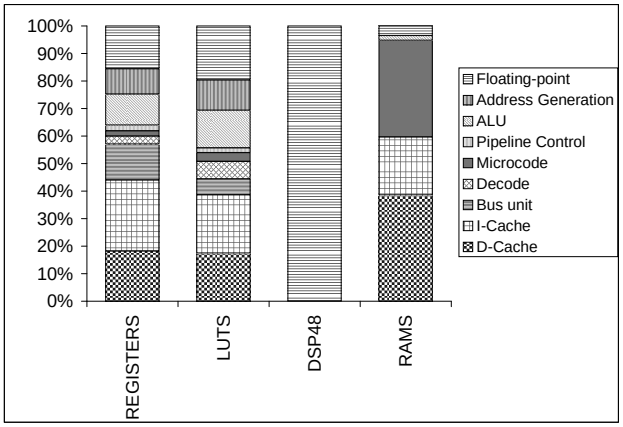

Figure 3: Breakdown of FPGA resources used by different parts of the Pentium® archtiecture.

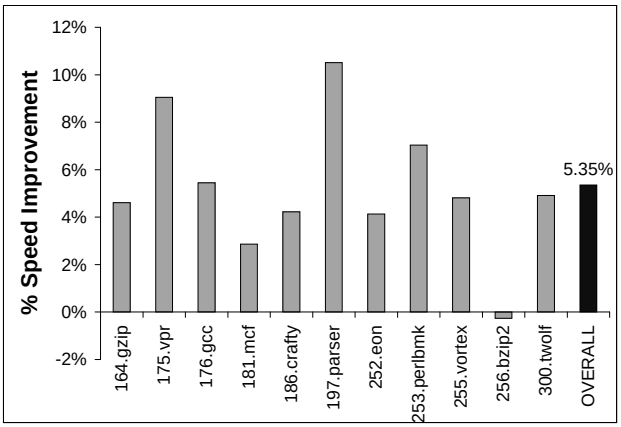

Figure 4: Performance increase of the doubled branch target buffer on SPEC2000 integer benchmarks.

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 [18] 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®. 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: Assuming modern 90nm desktop processors run up to 3.8GHz and have 5x the number of pipeline stages (and hence 5x the clock rate) we extrapolate and say that our Pentium® core would be clocked at 760 MHz in a 90nm process—approximately 30x faster than its 90nm FPGA counterpart. Although crude, the above analysis suggests that highly optimized transistor designs can perform multitudes faster than the expected 3x of a push-button FPGA flow.

With respect to area, we estimate that the number of transistors on the Virtex-4 LX200 is greater than 500 million. Since the 3.3 million transistor Pentium® used about 35% of these (we assume the number of transistors used is proportional to the LUT and BRAM usage in Table 1), that means the FPGA required 53x more transistors than the actual processor. Although this is also very crude and even coupled with the fact that transistor count is not an accurate measurement of area, the outcome agrees with our expectation of seeing higher overheads since the previously published results used a synthesis-based standard cell flow without manual optimization.

5.2 Comparing Branch Target Buffer Sizes

Doubling the branch target buffer should give the processor twice the 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 real speed improvements remain hidden in the noise inherent in the real system.

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 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 [11]. This results in an execution speedup of 27x for our custom crypto-engine versus the best software implementation. Table 2 summarizes the resource utilization of the AES and DES engines on the Virtex-4 FPGA

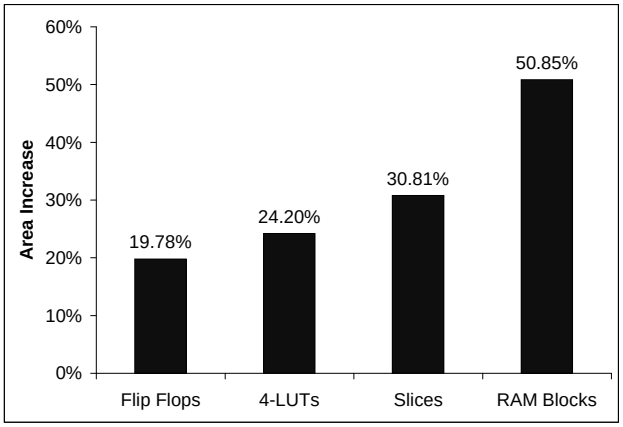

Figure 5: Area increase of the 32KB 8-way L1 caches versus the 8KB 2-way L1 caches.

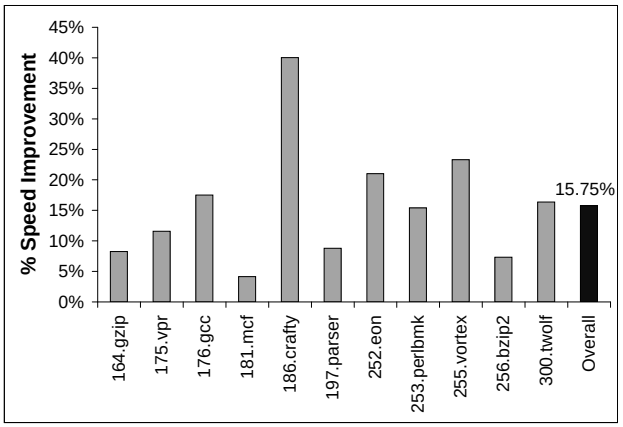

Figure 6: Performance increase of the 32KB 8-way L1 caches versus the 8KB 2-way L1 caches.

and shows that the logic requirement is very small but a substantial number of BRAMs were required. Nonetheless, for secure environments the extra resources would be well worth the performance improvement.

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 achieve newer heights of efficiency by optimizing across the entire system stack: architecture, instruction-set device drivers, operating systems, and applications without the inhibitive 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.

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 except for a significant need for block RAMs.

Table 2: Virtex-4 resource utilization of the AES and DES IP cores.

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

7. REFERENCES

[1] LEON SPARC. http://www.gaisler.com.

[2] Opencores.org. http://www.opencores.org.

[3] OpenSPARC. http://opensparc.sunsource.net/.

[4] PowerPC. http://www.power.org.

[5] T. Austin and D. Burger. The SimpleScalar Tool Set Version 3.0, 1998.

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

[7] D. Chiou, H. Sunjeliwala, D. Sunwoo, J. Xu, and N. Patil. 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*, 2006.

[8] P. S. M. et al. Simics: A Full System Simulation Platform. *IEEE Computer*, 35(2):50–58, 2002.

[9] G. Gibeling, A. Schultz, and K. Asanovic. RAMP: The RAMP Architecture and Description Language. Technical Report, 2006.

[10] G. Gibeling and J. Wawrzynek. A Universal Processor for RAMP. Technical Report, 2006.

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

[12] Hifn. 4450 HIPP III Storage Security Processor, 2006.

[13] Intel. The Pentium Datasheet, 1997.

[14] International Symposium on High-Performance Computer Architecture. *Workshop on Architecture Research using FPGA Platforms*, San Francisco, California, 2005.

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

[16] J. Kasper, R. Krashinksy, C. Batten, and K. Asanovic. 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*, 2005.

[17] C. Kozyrakis and K. Olukotun. 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*, 2005.

[18] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. In *FPGA '06: Proceedings of the 2006 international symposium on Field-programmable gate arrays*. ACM Press, 2006.

[19] S.-L. Lu, E. Nurvitadhi, J. Hong, and S. Larsen. 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*, 2005.

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

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

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