

Extensible Processors

Shawn J. Goff, *Student, IEEE*

One of the decisions faced by microprocessor designers is whether to implement a complicated instruction set computer (CISC) or a simpler reduced instruction set computer (RISC). A new area of study, extensible microprocessors, involves simplifying this decision. An extensible microprocessor has a RISC core with an integrated FPGA (field programmable gate array). The FPGA allows custom logic to be dynamically added to the system. While logic implemented in an FPGA is not as fast as dedicated logic in the microprocessor, it does allow very specific complicated instructions to be implemented. Because these instructions can be dynamically reprogrammed, complicated algorithms may be implemented in the FPGA instead of software. This provides significant speed enhancements. Several interesting applications have been discovered that take advantage of the extensible microprocessor.

Index Terms— extensible microprocessors, FPGA, VLSI

I. INTRODUCTION

IN THE FOLLOWING section, microprocessors and FPGAs are discussed. Following this background information, the extensible microprocessor will be introduced and interesting applications will be presented.

Microprocessors contain hardware logic units that perform very simple operations. Some examples of these operations include the following: move data between registers or from a register to memory, add or subtract two numbers, compare numbers, check a status register, and branch on a condition. A piece of software can execute these operations in a sequence to perform a more complicated operation. Complicated operations can also be implemented directly in hardware; it is possible to make a logic unit that will perform an arbitrarily complex operation. The hardware logic is able to execute these operations faster than their software equivalents because the operations take fewer steps to perform. More complicated operations yield greater speed advantages when implemented in hardware, but they will also be, in general, more specific than simpler instructions, so they will be used less. Additionally, since these operations could have been carried out on the simpler logic, the operations provide duplication of functionality that is already in the microprocessor. These instructions also increase the complexity of the system, making it more costly to design, optimize, and test. The hardware logic, once built into the system is not modifiable; it is fixed at design time.

These tradeoffs in implementing complicated instructions are things an engineer must consider when designing a microprocessor. The two opposing design paradigms are called *complicated instruction set computing* (CISC) and *reduced instruction set computing* (RISC). A RISC system implements a very basic set of instructions. A CISC system has more logic (and more instructions) dedicated to doing specific tasks, which means a CISC processor can do some tasks in fewer cycles than a RISC system. The decision of implementing CISC or RISC involves several tradeoffs that must be considered. A RISC system is easier to optimize than a CISC system. By using fewer steps, a CISC system can execute certain functions quicker than a RISC system, but because every logic device carries with it a timing delay, a

RISC system can run at faster speeds than a CISC system. RISC systems use less silicon, and hence are less costly, than CISC systems. Designers must find the sweet spot between implementing enough useful instructions while not overcomplicating the system.

FPGAs are hardware devices with many logic cells that can be interconnected in various ways to perform a designed operation. The logic cells in an FPGA are connected to large intersecting buses that have programmable switches at each intersection. The FPGA logic is dynamically configurable by changing which switches are closed and which are open. By connecting the logic devices in various ways, the designer can program whatever logic operations are necessary in the FPGA. The cycle speed of an FPGA, however, is much slower than a microprocessor. Due to this, an FPGA can be significantly slower than a microprocessor when performing a variety of general computing tasks. FPGAs excel with application-specific logic that saves several clock cycles over an equivalent microprocessor routine.

II. EXTENSIBLE MICROPROCESSORS

Combining a general purpose microprocessor and an FPGA yields a useful device: an extensible microprocessor. An extensible microprocessor includes advantages from both devices: a fast general purpose processor core that is simple and cost-effective, and programmable hardware logic for implementing complex instructions quickly.

Since the FPGA is dynamically programmable, each application or even each subroutine can have very specific logic instructions which can greatly speed up processing without needlessly complicating the processor with instructions that are not useful to other applications. A program intended for use on an extensible microprocessor includes any necessary hardware configuration as well as instructions that invoke the custom logic operations. There exist tools for some systems that allow the automatic patching of a program with relevant FPGA programming instructions. The normal software algorithms are also left in place as a fallback, in case the FPGA extension is not available.[4]

An important consideration when executing custom logic is that it must not provide an attack vector against other

programs. To guard against this, Sukhwani et. al. describe a secure execution environment where FPGA extensions are not allowed access to memory reserved for other applications. [5]

Much of the published research in extensible processors is done by Microsoft Research who have designed their own platform called eMIPS. eMIPS combines a Xilinx FPGA with a MIPS processor. In eMIPS, the FPGA has access to every stage of the datapath, allowing for significant flexibility and an accessible platform for extensions.

III. APPLICATIONS

Because of the unique advantages provided by extensible microprocessors, interesting applications have come about.

G. Busonera, et. al. discuss an on-chip debugging interface, eBug, for debugging tasks that would be either too expensive or too inflexible to implement in fixed hardware. [2] eBug watches the execution of programs passively – not interacting with them until an exception occurs. Since this logic is implemented in the FPGA, the software under test is completely unaffected by the presence of the debugger. In addition, the FPGA portion is reprogrammable, so when the debugger is not in use, that area can be reused for another purpose; this is in contrast with debugging interfaces implemented in fixed hardware, such as JTAG, which are common today.

B. Sukhwani introduces extensible I/O systems, which are capable of adding custom I/O functionality to a microprocessor. [6] Previously, it was only possible to add I/O functionality to an external chip that communicated to the microprocessor through a predefined interface. Customizable I/O opens up many possibilities for directly connecting the microprocessor to other devices to allow for more efficient, less latent communications that are easier to program.

A group of researchers from the University of Karlsruhe in Karlsruhe, Germany created an extensible processor that is self-adapting. The processor constantly analyzes the usage extended instructions as it runs and controls how and when the custom logic is configured. The team benchmarked their system by encoding a video using the H.264 algorithm. Compared to a traditional embedded system without hardware accelerators, they obtained a speedup of 10.5x to 17.5x. [1]

Hong Lu and Alessandro Forin demonstrate a zero-overhead online software validation. Software validations helps to ensure the correctness of software – not just that it doesn't contain bugs, but that it does what the designers claim and what the client has requested. By using the FPGA attached to a microprocessor, Lu and Forin were able to implement assertion-based validation that does not alter the behavior of the program under test in any way. [3] This is especially important for validating either timing-based or performance-critical assertions.

IV. CONCLUSION

Extensible processors offer the advantages of being able to execute complicated instructions without overcomplicating a processor design or using extra space on the semiconductor

wafer. Instead of implementing all logic in the silicon, the custom logic instructions can be dynamically modified because they are implemented in a reprogrammable FPGA. The subject of extensible processors is very nascent, but the applications already made possible are very interesting. The addition of configurable logic to a microprocessor should also provide significant speed boosts for complex algorithms.

REFERENCES

- [1] L. Bauer, M. Shafique, D. Teufel, and J. Henkel, "A Self-Adaptive Extensible Embedded Processor," *First International Conference on Self-Adaptive and Self-Organizing Systems*, 9-11 July 2007, pp. 344 - 350
- [2] G. Busonera, A. Forin, R. N. Pittman, "Exploiting Partial Reconfiguration for Flexible Software Debugging," *Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008. SAMOS 2008. International*, 21-24 July. 2008, pp. 173-181
- [3] H. Lu and A. Forin, "Automatic Processor Customization for Zero-Overhead Online Software Verification," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. Vol. 16, Issue 10, Oct 2008, pp. 1346 – 1357.
- [4] A. Sekar and A. Forin, "Automatic Generation of Interrupt-Aware Hardware Accelerators with the M2V Compiler," *Microsoft Research MSR-TR-2008-110*, Microsoft Research, WA, August 2008.
- [5] B Sukhwani, A. Forin, R. N. Pittman, "Extensible On-Chip Peripherals," *2008 Symposium on Application Specific Embedded Processors*, 8-9 June 2008, pp. 55-62.
- [6] B Sukhwani, A. Forin, R. N. Pittman, "An Extensible I/O Subsystem", *16th Annual Symposium on Field-Programmable Custom Computing Machines*, 14-15 April 2008, pp. 269-270.