

Configurable Soft Processor Arrays Using the OpenFire Processor

Stephen Craven, Cameron Patterson, and Peter Athanas
Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University, Blacksburg, VA 24061
{scraven | cdp | athanas@vt.edu }

Abstract

Single-chip multiprocessor systems, while requiring significantly less design effort than custom hardware solutions, fall behind custom RTL in performance. In an effort to decrease this performance gap, the individual processors in an array can be tailored to their specific application. In this paper we present the OpenFire, a Xilinx MicroBlaze-compatible processor designed for configurable array research. A sample application is presented demonstrating the benefits of configurable processor arrays. By sizing the datapath width to better suit an image filtering application the OpenFire processor is made smaller, allowing additional processors to be added to the array, thus increasing the overall throughput.

I. Introduction

Increasing interconnect delay in smaller process technologies has led microprocessor developers to place multiple processors on the same die to increase performance instead of using the additional transistors for architectural improvements. All major processor vendors, including AMD, IBM, and Intel, produce chips with dual processor cores. Next generation gaming consoles from Microsoft and Sony sport chips with three and eight processing units, respectively. To reap power savings while obtaining the necessary performance mobile devices typically pair a microcontroller with a signal processor.

Today's deep sub-micron technologies allow for hundreds of small processors to be placed on the same chip. The PC102 from picoChip contains 308 RISC processors in an array with an additional 14 co-processors [7]. It has been calculated that 3,000 Arc600 32-bit RISC processor cores can fit on a single die, producing 600,000 MIPS while consuming only 25,000 MIPS / watt [1]. By reusing previously implemented processors, design and verification costs are significantly reduced resulting in faster time-to-market and decreased overall risk. Furthermore, the shorter maximum wire lengths within a single processor allow each processor in the array to run faster than a single processor occupying the same die area.

While the benefits of processor arrays are impressive, they are not free. Programming for a multiprocessor system is more difficult than for a single processor. System performance on single threaded tasks is reduced. Furthermore, for a specific application a custom hardware design would produce noticeably better performance compared to a processor array of the same die area. While the increased design time and resource requirements make custom hardware solutions unattractive for many applications, it is possible to reap some of the gains of custom hardware in multiprocessor systems by configuring each processor to its specific task.

Application-Specific Instruction set Processors (ASIPs) are processors tailored to a specific application or set of applications through instruction set and architectural modifications.

Previous research has demonstrated that an ASIP offers significant speed-ups over a generic processor [3] [4]. Automated flows, such as ASIP Meister and the Tensilica design flow, have been developed that examine an application written in a high level language and customize the processor for the specific task. Tensilica's tools perform three types of modification on their Xtensa processor: instruction set extension through instruction fusion, multiple instruction issues using VLIW instructions, and parallel data operations from SIMD instructions.

Tensilica and ASIP Meister are powerful tools, but each has limitations. Both projects, while producing synthesizable HDL code, are tailored for ASICs. Even though ASIP Meister is freely available for download, it is closed source and does not provide a command line interface for use in an automated flow. Tensilica starts with a basic processor and adds extensions, possibly burdening the final design with unused functionality of the basic design. Finally, neither project allows exploration of alternative processor structures such as, for example, the replacement of a portion of the register file with a FIFO for faster access and lower area.

Contemporary research into multiprocessor systems has placed considerable importance on the interconnection between the processors. The higher the communications bandwidth the more applications can take advantage of parallel implementations. MIT's RAW processor [5] utilizes almost half of the die area for different types of networks. For many applications including signal processing, however, the communications patterns are very regular with processors only needing fast communication to other near-by nodes. In such cases the overhead of a complex networking scheme will consume area that could be better used for additional processors. As such we propose a simple and constant communication interface consisting of FIFOs between nodes. This interconnection network allows data to travel from a register on the source processor to a register in a neighboring node in two clock cycles.

As a stepping-stone for our research into optimizing arrays of soft processors, we have created a configurable 32-bit RISC processor that leverages the development tools of the Xilinx MicroBlaze soft processor. Our OpenFire processor executes a subset of the MicroBlaze instruction set, eliminating instructions and functionality that are not needed in a processor array.

This paper is organized as follows. Section II describes the OpenFire processor and associated tools. In Section III we present results of a sample OpenFire application, median image filtering, optimized through datapath sizing. The future directions presented in Section IV focus on defining a methodology for creating application-specific processor arrays. Finally, conclusions are presented in Section V.

II. OpenFire Configurable Processor

For our investigations into configurable processor arrays a suitable processor was needed. A comparison of the OpenRISC, Leon, and MicroBlaze soft processors found that the Xilinx MicroBlaze has the highest performance per slice [2]. The MicroBlaze is based off of Hennessy and Patterson's DLX RISC architecture and has been designed for easy implementation and high performance. Because of its high performance, extensive tool support, and support for a simple interface, the Xilinx MicroBlaze was selected.

As Xilinx has not released the source code for the MicroBlaze, instead requiring users to purchase and use their Embedded Development Kit (EDK), open source alternatives were explored. An OpenCores project, aeMB, purports to be a clean-room implementation of the MicroBlaze instruction set [8]. After some experimentation it was decided not to pursue aeMB because of lack of documentation, incorrect simulation results, and the absence of needed instructions such as compare and multiply. Given the simple pipeline and instruction set of the MicroBlaze and the unsuitability of the only other open source implementation, it was decided to create a fresh implementation including only the functionality needed in a processor array.

Our implementation of a subset of the MicroBlaze instruction set is called the OpenFire. Featuring the same three-stage pipeline (fetch, decode, and execute) of the MicroBlaze, the OpenFire is cycle-accurate with the MicroBlaze for most instructions. The entire MicroBlaze version 3 instruction set is implemented with the following exceptions: barrel shifting, hardware division, status register related instructions, and cache related instructions are not implemented. Furthermore, exceptions and interrupts are not currently implemented. This extensive support for the MicroBlaze instruction set allows the OpenFire to execute most code compiled in the MicroBlaze tool chain using mb-gcc. The OpenFire is cycle-accurate to the MicroBlaze for every implemented instruction with a single exception. The hardware multiplier on the OpenFire introduces five cycles of latency as opposed to three cycles in the MicroBlaze.

Further differences between the MicroBlaze and OpenFire are apparent in interfaces. The MicroBlaze supports three bus types: Fast Simplex Link (FSL), Local Memory Bus (LMB), and the On-Chip Peripheral Bus (OPB). The LMB is used as a high-speed connection to on-chip block memory. The OPB connects to a wide assortment of Xilinx IP cores and the FSL is designed as a simple and fast FIFO-like interface to on-chip co-processors. To simplify implementation and reduce area, the OpenFire only implements the FSL interface for use as the interconnect network in the processor array. The LMB interface, while providing a simple interface to memory, limits the minimum memory size of an implementation to 8 kB on a Xilinx Virtex-II. The LMB would further complicate OpenFire's design by requiring two cycles to access instruction memory. To ensure a single-cycle fetch stage, the MicroBlaze implements a pre-fetch buffer to hide the LMB latency. Instead of using the LMB, the OpenFire uses a simple single-cycle connection to local memory, alleviating the need for the pre-fetch buffer. As the OpenFire does not feature an interface to external memory, a cache is not implemented, further reducing area.

Performance of the OpenFire is comparable to the MicroBlaze in Dhrystone MIPS and area, as shown in Table 1, below. The OpenFire's maximum speed in a xc2vp30-6 is 100 MHz using a 32-bit datapath. When compared to a minimal MicroBlaze implementation (no OPB, cache, barrel shifter or hardware divider) the OpenFire is 13% smaller and less than 1% slower in version 2.1 of the Dhrystone benchmark. The slight difference in DMIPS is most probably due to the OpenFire's longer multiply latency.

Table 1: Performance Comparison (100 MHz, xc2vp30)

	Area (slices)	DMIPS
OpenFire	641	58.47
MicroBlaze*	734	58.98

* Minimal MicroBlaze implementation

To further reduce the area and increase the throughput, the datapath width of the OpenFire can be varied. Currently implemented values include 32-bits and 16-bits, although any width can be easily supported. At a datapath width of 16-bits, the area reduces to 402 slices, a 37% reduction from the 32-bit OpenFire and a 45% reduction from a MicroBlaze. Code generated from the MicroBlaze compiler can function without modification on the 16-bit datapath assuming, of course, that the code only uses variables that fit into 2 bytes. The standard string functions for C, however, do not work with a different datapath size. Care should be taken to ensure that the code does not make use of the higher order bytes.

While datapath sizing can save significant area, there is a drawback. Because the OpenFire register file consists of only general-purpose registers in which to store addresses, the maximum addressable location in memory is limited by the datapath width. For example, for a datapath width of 8 bits, the maximum address value that can be stored in a register is 0xFC, corresponding to a maximum address space of 64 words. Given that most programs require more locations than this, a higher datapath width is needed. For a datapath width of 16-bits, the addressable memory size is 16 kB.

Many of the Xilinx MicroBlaze tools can be used with OpenFire. The Xilinx Platform Studio (XPS) recognizes the OpenFire and can be used to connect the OpenFire to other peripherals through the FSL interface. The MicroBlaze compiler, mb-gcc, and associated tools generate and analyze executables for the OpenFire. Because the OpenFire does not use the LMB or OPB interfaces to connect to memory, however, the Xilinx EDK tools cannot be used to load OpenFire programs into memory. A simple script has been written to determine the placed locations of an OpenFire's memory and populate these with the executable. For verification purposes a simple C simulator has also been created that produces log statements in the same format as OpenFire's top-level Verilog test bench. This allows easy verification of the RTL by simply comparing the Verilog simulator's output to that of the C simulator.

OpenFire is released under the MIT license, allowing free and unrestricted use of the source code. The current release, version 0.2, is limited to a single FSL interface, allowing ring network topologies but prohibiting 2-D connections.

III. Image Filtering Application

As an initial exploration of configurable processor arrays, a 3x3 median image filter was created in C. This filter accepts images in Windows Bitmap format using 8-bit color. To map the filter to an array of processors the image is divided into square blocks of 64 pixels width. A master processing node distributes a block to each slave processor, after which the slave nodes filter their blocks and return the result. A ring network is formed, as shown below in Figure 1, using the

FSL interfaces. The master MicroBlaze is needed because the OpenFire does not provide OPB support needed to communicate with the external DDR SDRAM.

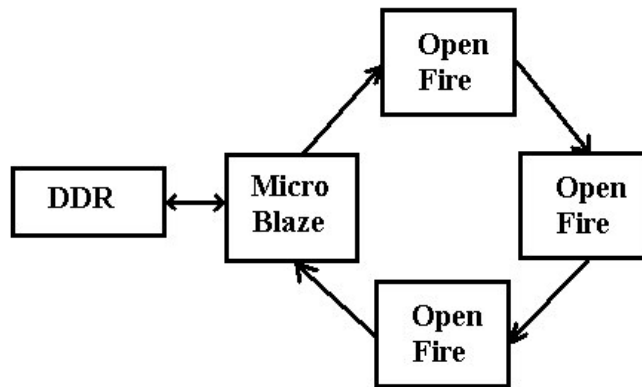


Figure 1: Image Filtering Multiprocessor System

The master MicroBlaze does not perform any filtering and instead acts merely as a memory interface and network controller. The MicroBlaze initializes the network by first transmitting the number of nodes in the network to the first OpenFire. The slave node, upon receiving the number of nodes, decrements the number before retransmitting. This syncs each node to the master and allows each node to know its location in the network. The master node then parses the image in external memory, dividing the image into 64x64 pixel blocks. Each node is then sent a block, after which filtering is performed. The heart of the filtering algorithm is the insertion sort algorithm, which makes heavy use of the OpenFire's optional comparator. After filtering is completed, the slave nodes transmit the blocks back to the master MicroBlaze, which writes the results to memory before sending new unfiltered blocks to the slave nodes. In addition to its interface and control tasks, the master MicroBlaze records the running time of the entire program using a timer peripheral.

Near linear speed-up for this application was observed as the number of nodes were varied. As Figure 2 shows, performance was slightly lower (15%) than a system with linear speedup, as is expected due to the increasing communication overhead with each node added.

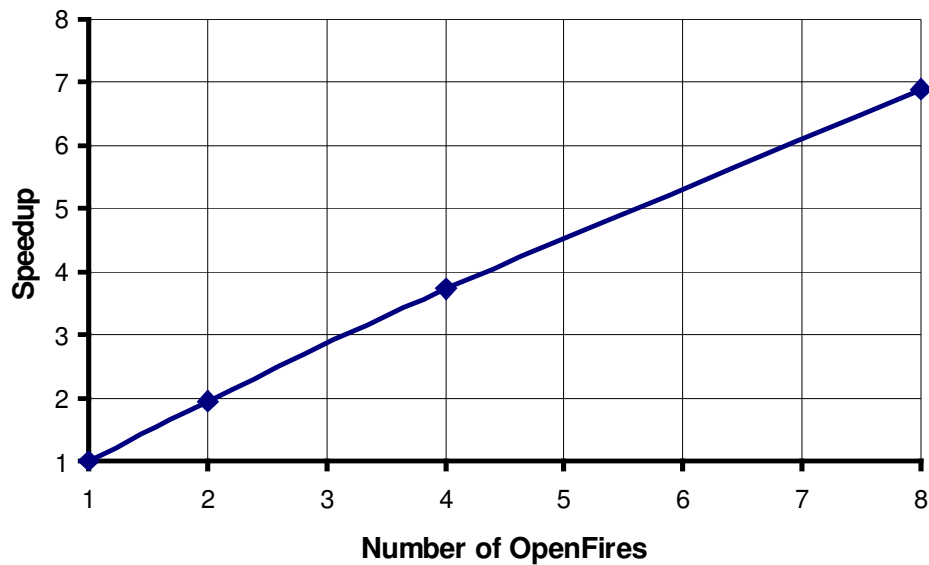


Figure 2: Image Filter Speedup versus Number of Processors

IV. Future Directions

The OpenFire was created to facilitate research into configurable processor arrays. The eventual goal of our efforts is a design flow to automatically implement an application-specific processor array. Our initial investigations focus on a design methodology like that in Figure 3, below. The input to the design flow is a description of the application written as a parallel C program. Preferably the critical portions of the code are written as scalable programs, as was our image filtering application, so that these portions can be implemented in additional processors to improve overall performance. In this way the application can take advantage of two types of optimizations, speed improvements and area reductions, to improve the throughput of the entire array.

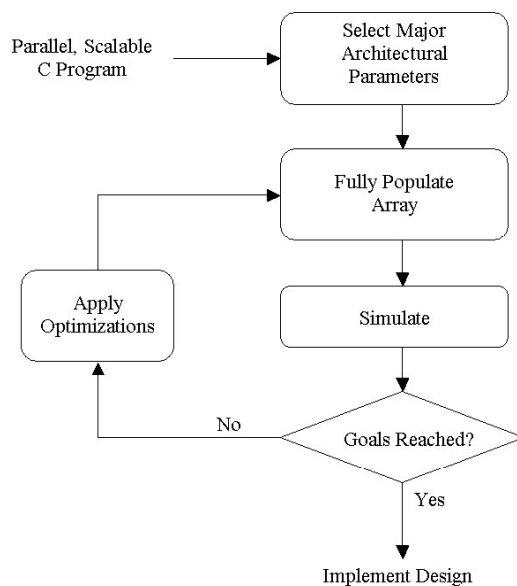


Figure 3: Preliminary Design Methodology

With an appropriate C program as input, the designer first selects the major architectural parameters, such as memory size and datapath width. The available chip area is then populated with a homogeneous array of processors and the application mapped onto the array using the FSL busses as interconnects. After simulation, if the performance goals have not been met, optimizations are applied to the individual processors. These optimizations, discussed below, aim to either increase the throughput of the individual processor or decrease its area. Any regained area is then used to implement more processors for critical program tasks, such that the entire array's performance is increased. The process is repeated until performance goals are met or no additional automated optimizations exist. If additional performance is needed beyond what the automation provides, the designer can port the critical sections directly to custom hardware using an appropriate tool, such as Xilinx's System Generator [6].

Optimizations under consideration include custom datapath widths, instruction set reduction and extension, direct ALU-to-FSL and FSL-to-ALU connections, register file extension and reduction, multiple instruction issues (VLIW), multiple data memories, and microcode controllers. These optimizations aim to reduce processor array, increase processor throughput, or improve inter-processor communication. In all cases the processor interfaces remain the same. This constant simple communication interface greatly eases debugging and optimization. As the FSL interface is both hardware and software friendly, a wide range of optimizations, including replacing the processor with a pure hardware datapath, can be performed without changing the interface.

V. Conclusion

Application performance can be improved by tailoring the processor to the specific application. For applications conducive to a parallel implementation, an array of custom processors offers

many benefits over a custom hardware implementation including reduced design costs and time-to-market improvements. The OpenFire processor was designed for just such applications and is unburdened by interfaces and hardware not used in processor arrays. Based on the Xilinx MicroBlaze, the OpenFire makes use of the mature MicroBlaze toolset while featuring configurability and extensibility not available in the MicroBlaze. Implementation results show for certain applications reducing the datapath width by half can decrease processor area by 37%, allowing more processors to be placed in the array. For a median image filtering application, near linear speed-up was observed. Work is continuing in the creation of an automated design methodology for generating application-specific processor arrays.

References

- [1] CMPWare, Inc., <http://www.cmpware.com/>.
- [2] Mattsson, D., and Christensson, M., "Evaluation of Synthesizable CPU Cores", Master's Thesis, Chalmers University of Technology, 2004.
- [3] Rowen, C., Engineering the Complex SOC, Prentice Hall, Upper Saddle River, New Jersey, 2004.
- [4] Sun, F., Srivaths, R., Raghunathan, A, and Jha, N., "Synthesis of Application-Specific Heterogeneous Multiprocessor Architectures Using Extensible Processors", Proceedings of 18th International Conference on VLSI Design, Kolkata, India, Jan. 3-7, 2005.
- [5] Taylor, M., et al., "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs", IEEE Micro, vol. 22, no. 2, Apr. 2002, 25-35.
- [6] Xilinx, Inc., "Xilinx System Generator v7.1 User Guide", Available for download at: http://www.xilinx.com/products/software/sysgen/app_docs/user_guide.htm.
- [7] <http://www.picochip.com>
- [8] <http://www.opencores.com/aemb>