A VLIW Processor with Hardware Functions: Increasing Performance While Reducing Power

Raymond Hoare, Alex K. Jones, Dara Kusic, Joshua Fazekas, Gayatri Mehta and John Foster
Department of Electrical and Computer Engineering, University of Pittsburgh
{hoare, akjones}@ece.pitt.edu

Abstract

This paper introduces an embedded hybrid processor that increases performance by more than an order of magnitude while also reducing power consumption by a similar amount, even at the higher performance. This is achieved by moving key software functions into hardware through an automated process. These functions are labeled *hardware functions* and are controlled by the software processor. Specialized dual-ported memories are used to enable data sharing between the software processor and the hardware functions so that there is no overhead associated with calling the hardware functions. Adding hardware functions resulted in dramatically improved performance and reduced energy consumption.

Our processor has been synthesized for 160nm standard cell ASIC fabrication process from OKI and for a 90nm Stratix II FPGA with a core operating frequency of 167 MHz for both technologies. We present multimedia and signal processing benchmarks that show kernel performance improvements over a single processor ranging from 9X to 332X, and entire application speedups ranging from 4X to 127X. Hardware functions also provide many orders of magnitude of power improvement for the computational kernels, ranging from 42X to over 418X.

Introduction

The work described herein studies the performance gains, power and energy savings of a heterogeneous multi-core embedded processor that combines a very-long instruction word (VLIW) processor with application-specific, combinational-logic hardware functions. The constituent cores share a register file without a bus and associated overhead. C-code benchmarks from the Mediabench set have been selected to test the architecture. The addition of hardware functions lends power and performance improvements due three main effects: (1) *predicated control flow* (2) *cycle compression* and (3) *power compression*. Dedicated hardware lends efficiency to branching structures and specialized computations that become *predicated control flow* and *cycle compression*. *Power compression* is an effect from reducing the functionality from a general purpose arithmetic logic unit (ALU) to application specific functional units. This idea is expanded to take advantage of power-optimized, general purpose ALUs.

This work partially supported by the Swanson Center for Micro and Nano Systems and the Technology Collaborative.

Architecture Description

In order to provide a realistic baseline processor design, we selected the 32-bit Nios II processor from the Altera Corporation that targets Altera devices. Our processor has an identical instruction set architecture but has been custom-designed to seamlessly integrate to the hardware functions. We extended the processor core to a 6-stage pipelined VLIW processor with four fully functional processing elements. The architectural overview is shown in Figure 1 and is described in greater detail in [1], [2].

We selected the Altera Stratix II EP2S180F1508C4 FPGA with a maximum internal clock rate of 420 MHz as our target device. The EP2S180F has 768 9-bit embedded multiply-accumulate ASIC cells and 1.2 MB of internal memory. After iterative optimizations to the critical path, the clock rate reached its present 4-wide VLIW rate of 167MHz.

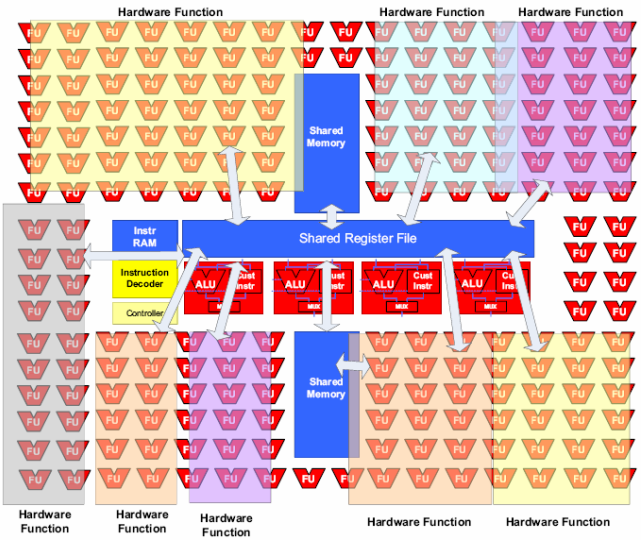
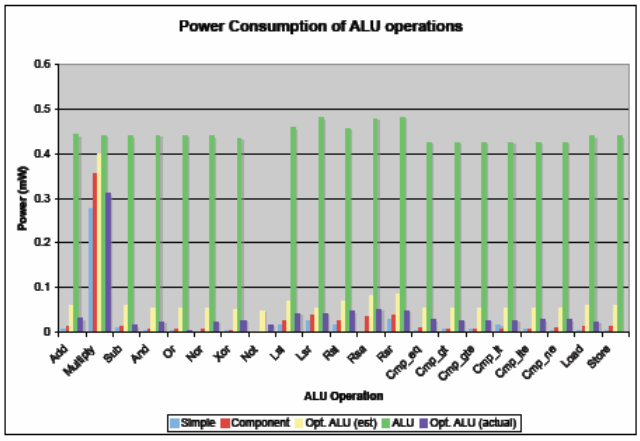


Figure 1: VLIW processor architecture with application specific, combinational hardware functions.

To achieve dramatic performance increases, we observe that hardware typically executes over 10x faster than software for the same functionality using the same fabrication technology. We also observe that computational portions of a typical processor consume a very small percentage of the processor area and an even smaller portion of an FPGA or ASIC. It is not possible to move entire software applications into hardware but fortunately, the 90/10 rule states that 90% of a programs execution is spent in only 10% of its code. Thus, we move the performance-critical kernels into hardware and execute the remaining code in software.

Through profiling, the software kernels are identified and converted into *hardware functions.* The hardware functions are strictly combinational logic and do not contain any internal data storage. Data is stored in the shared register file or in shared memory, accessible to both the processor and the hardware functions. The program execution is controlled by a shared *instruction decoder* that orders the execution of both software and hardware.



In software, an *if-then-else* statement is implemented in no less than six assembly instructions and stalls the processor pipeline when branch prediction misses. By converting forward-branching control flow into hardware functions, we increase the size of the hardware functions and dramatically improve performance. Thus, complex decisions execute using efficient combinational hardware rather than inefficient software branching. Push-button hardware synthesis tools produced hardware functions that performed significantly faster than we estimated.

Two additional benefits of the combinational hardware functions are *cycle compression* and *power compression.* All software instructions consume one or more cycles. However, from our experiments, the average computation requires less than 40% of cycle time when implemented in hardware. Similarly, the power consumption of hardware compared to software for the same operation is much less, a concept we call power compression. Figure 2 shows power consumption for various arithmetic logic unit (ALU) designs with simplest hardware units listed as Components.

Results

All of our Mediabench benchmarks showed significantly limited instruction level parallelism (e.g. < 2) for both kernel and non-kernel portions of the code [3]. Thus, the "ideal" software speedup was < 2 but by moving the software kernels into hardware functions, we achieved gains that exceed parallelism. The compilation flow targeting our architecture is described in detail in [1], [2]. For the FPGA-based implementation, we used Synplify Pro synthesis with Quartus II place-and-route software. For ASIC implementation, we used Design Compiler synthesis for 160 nm OKI standard cells. Functional simulations were done in ModelSim and power estimations for the ASIC implementation were generated from PrimePower.

Table 1 provides a summary of the performance results of a variety of architectural choices. Hardware functions provides nearly 10X to over 200X speedups over a single processor implementation for application speedups ranging nearly 3X to over 25X [2]. Due to limited ILP, the 4-way processor provides only nominal speedups ranging from 1.02X to 1.48X. An unlimited-way VLIW is able to achieve only slight improvements over the 4-way VLIW. However, when the VLIW processor is combined with hardware functions, the speedup improves to a range from 4X to over 30X. The energy and power required for a computation kernel to complete within a hardware function compared to the VLIW processor is shown in Table 2. Results show a power improvement ranging from just under 50X to over 400X, with an average improvement of 130X.

Table 2: Performance results

	HW Function		VLIW-4	
	Power(mW)	Energy(μJ)	Power(mW)	Energy(μJ)
ADPCM Dec	7.53	1.16	701.2	2379
ADPCM Enc	8.59	3.69	695.6	6265
IDCT Row	6.47	2.96	708.7	43702
IDCT Col	13.34	13.07	708.5	43702
G.721	16.71	256.37	701.8	3442614
GSM	1.69	101.66	705.2	421131

In conclusion, the results of our test show that by moving software kernels into hardware functions, we gain between 9X and 332X in performance for the kernels and between 4X and 127X in performance for the entire benchmark. Given standard cell implementation, power reduction from 42X to 418X is achieved with average savings of 133X.

Table 1: Benchmark kernel performance summary

	Kernel				Application					
	Seq	VLIW-4	VLIW-Unl	HW	Seq	VLIW-4	VLIW-Unl	Seq+HW	VLIW-4+HW	VLIW-Unl+HW
ADPCM Encode	1.00	1.14	1.14	18.25	1.00	1.28	1.28	4.00	7.66	7.66
ADPCM Decode	1.00	1.02	1.02	18.33	1.00	1.13	1.13	2.92	4.15	4.15
G.721 Decode	1.00	1.25	1.42	230.00	1.00	1.27	1.39	26.49	31.50	31.50
GSM Decode	1.00	1.33	1.33	9.33	1.00	1.35	1.35	4.13	5.81	5.81
MPEG2 Decode	1.00	1.37	1.39	51.14	1.00	1.46	1.55	6.99	8.82	11.73
Spherical Decoder	1.00	2.68	2.70	332.50	1.00	2.66	2.68	127.29	127.29	127.29

References

[1] R. Hoare, A.K. Jones, et al, "Rapid VLIW Processor Customization for Signal Processing Applications Using Combinational Hardware Functions," *EURASIP Journal on Applied Signal Processing*, 2005, in second review.

[2] A. K. Jones, R. Hoare, D. Kusic, J. Fazekas, , and J. Foster, "An FPGA-based VLIW Processor with Custom Hardware Execution," in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2005.

[3] D. Kusic, R. Hoare, A. K. Jones, et al, "Extracting Speedup from C-code with Poor Instruction-level Parallelism," in *Workshop of Massively Parallel Processing (WMPP)*, 2005.