OPPORTUNITIES FOR EMBEDDED SOFTWARE POWER REDUCTIONS

Omid Assare[1], *Maziar Goudarzi*[2]

[1]Electrical Engineering Department, Sharif University of Technology, omid@ee.sharif.edu
[2]Computer Engineering Department, Sharif University of Technology, goudarzi@sharif.edu

ABSTRACT

While performance and power consumption of processors present a classic trade-off in designing embedded hardware, software can be optimized in favor of both performance and energy. We evaluate the impact of optimizations at different stages of designing embedded software. We show that algorithm choice and compiler optimizations aimed at improving performance can also reduce energy consumption of an embedded processor. We also propose energy-aware compilation guidelines which can further reduce energy consumption without performance penalties. Our experimental results show that up to 85% energy reduction and 89% performance improvement can be achieved by these techniques.

***Index Terms*—** Low-power software, energy-aware compilation, embedded systems

1. INTRODUCTION

Widespread use of portable electronic devices has altered the traditional performance-oriented design process of embedded computer systems. Both hardware and software designers must now have deep knowledge of their decisions on the final product's energy consumption. Unlike hardware design in which power or performance are usually inhibited in favor of the other, performance-improving techniques in software will also result in energy savings most of the time. Among software techniques for energy reduction, the ones that target high-level source code are favorable as they have a fundamental effect and can lead to significant energy savings.

In this paper, we will analyze the power consumption of software on embedded processor cores. Specifically, we will evaluate the effect of early algorithmic decisions as well as compiler optimizations on performance and energy consumption improvements. A number of techniques for energy-aware compilation are also presented. It should be noted that, unless where stated, the terms *power* and *energy* refer to dynamic power and energy, caused by switching activities of circuit nodes.

This work is supported in part by Iran National Elite Foundation.

CCECE 2011 Niagara Falls, Canada
978-1-4244-9789-8/11/$26 ©2011 IEEE

The rest of the paper is organized as follows. Section 2 reviews previous attempts for power reduction. Section 3 describes the experimental methods used in this work. Sections 4 and 5 analyze the effects of algorithm choice and compiler optimizations on performance and energy consumption of embedded systems. In section 6 we present two techniques for energy-aware optimization of the compiled code. Finally, conclusion and future work are explained in section 7.

2. RELATED WORK

Software-oriented techniques for power reduction can be roughly split into two categories: those targeting high-level source code and those that target assembly code and compiler techniques. Pedram presented a good review of techniques and tools for power-efficient embedded system design, considering the hardware platform, the application software, and the system software [1]. Simunic et al. [2] used algorithmic optimizations for MP3 decoding. Dalal and Ravikumar [3] analyzed a number of high-level optimization techniques such as loop unrolling and function inlining. Ortiz and Santiago [4] evaluated the impact of several optimization techniques on energy consumption of micro-processor based systems.

Low-level code optimizations were first proposed by Tiwari et al [5]. They exploited global register allocation of temporaries and frequently used variables and could reduce energy consumption of a small piece of code by 33%. Considerable amounts of research have been conducted on scheduling techniques to reduce power consumption [6 7 8]. The main focus of these has been on limiting dynamic power by reducing the switching activity of processor nodes.

3. EXPERIMENTAL ENVIROMENT

OpenRISC 1200 [9] was used as the microprocessor under study. OpenRISC 1200 is an implementation of the OpenRISC 1000 microprocessor family. It is a synthesizable microprocessor designed, developed and managed by OpenCores. The OpenRISC 1200 is a 32-bit pipeline RISC microprocessor with Harvard architecture. It targets medium and high performance networking, embedded, automotive and portable computer environments. Being an open-core

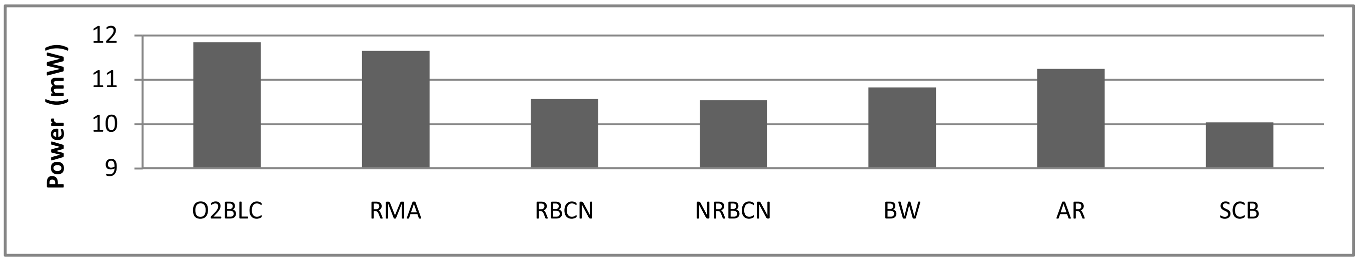

Figure 1-Average Power Consumption of bitcount Functions

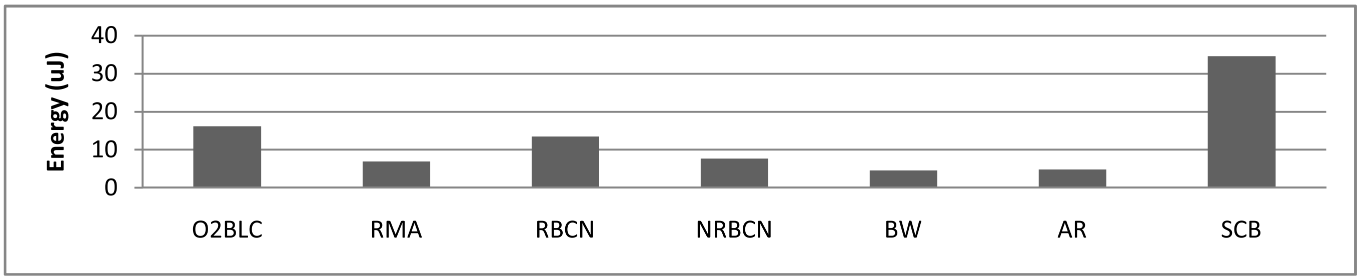

Figure 2-Energy Consumption of bitcount Functions

microprocessor, OpenRISC enabled us to perform detailed analysis on its performance and power consumption using low-level simulations. As a minimal system, a memory controller to interface with off-chip memory and a UART controller to serve as system output are connected to the processor core via a WISHBONE bus. These are integrated in a top module and synthesized, placed and routed on Cyclone III EP3C5E144C7 FPGA using Quartus II. 90% of the device logic elements were utilized in the design and Quartus reported a maximum frequency of 58.55MHz.

As the software, we used MiBench benchmark suite [10]. MiBench is a publicly available benchmark suite representative for several embedded system domains. To study algorithm choices, *bitcount* benchmark was chosen as it consists of seven functions performing the same job utilizing different approaches, enabling us to analyze the effect of algorithm choice on performance and power consumption. Four other benchmarks of MiBench, namely *qsort, basicmath, stringsearch,* and *fft,* were also executed on the processor to identify power-hungry components of the core, which further prompted us to propose energy-aware compilation guidelines.

Altera's PowerPlay Power Analyzer was used to obtain power consumption estimates. All the power consumption estimations reported in this paper have been done using full post-fit netlist simulation in ModelSim ALTERA and with the highest possible confidence metrics possible: more than 99% of the toggle rate data in all estimations were derived from simulation. PowerPlay uses signal activity information, namely toggle rate and static probability, contained in the VCD file generated by ModelSim along with capacitance models of all the nodes in the design to accurately estimate power consumption.

4. EFFECT OF ALGORITHM CHOICE

In this section, we investigate the importance of algorithm choice in designing energy-aware embedded software. As stated earlier, we used *bitcount* benchmark functions as they use seven different algorithms to perform the same job, counting the number of bits set in an integer input. This provides a fair framework for judging algorithms in terms of performance and energy consumption. These functions are introduced below:

- *Optimized one bit/loop counter (O2BLC)* uses a do-while loop which executes once for each bit set.
- *Ratko's mystery algorithm (RMA)* does not use any loops; it uses a five stage AND-SHIFT-ADD algorithm to count the number of bits set.
- *Recursive bit count by nibbles (RBCN)* is a recursive function counting the number of bits nibble by nibble.
- *Non-recursive bit count by nibbles (NRBCN)* uses a lookup table instead of recursion.
- *Non-recursive bit count by bytes by (BW)* and *non-recursive bit count by bytes by (AR)* are two look-up table based functions counting bits byte by byte.
- *Shift and count bits (SCB)* uses the basic algorithm of shifting each bit out and checking if it is a one.

Figures 1 and 2 show average power and energy consumption of these functions respectively. Two important conclusions can be drawn from these results:

First, it is clear, by comparison of these figures, that the dominant factor in determining a function's energy consumption is its run-time, not average power consumption of its instructions. For instance, while *shift and count bits* average power is 15% lower than that of *optimized one*

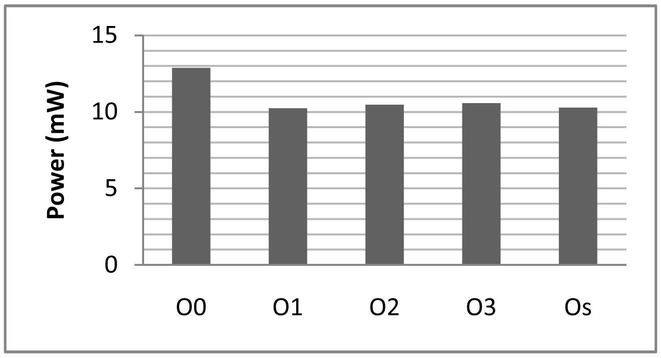

Figure 3-Average Power Consumption of *recursive bitcount by nibbles* with Different Levels of Optimization

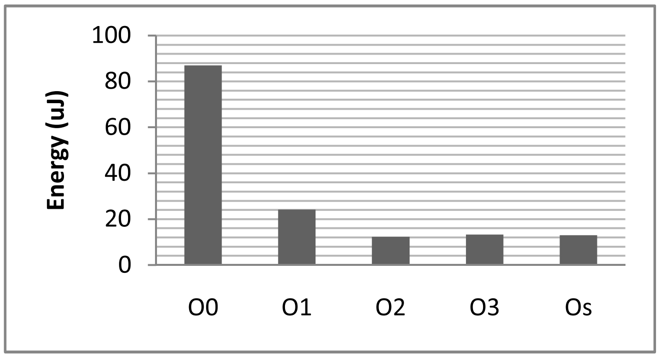

Figure 4-Energy Consumption of *recursive bitcount by nibbles* with Different Levels of Optimization

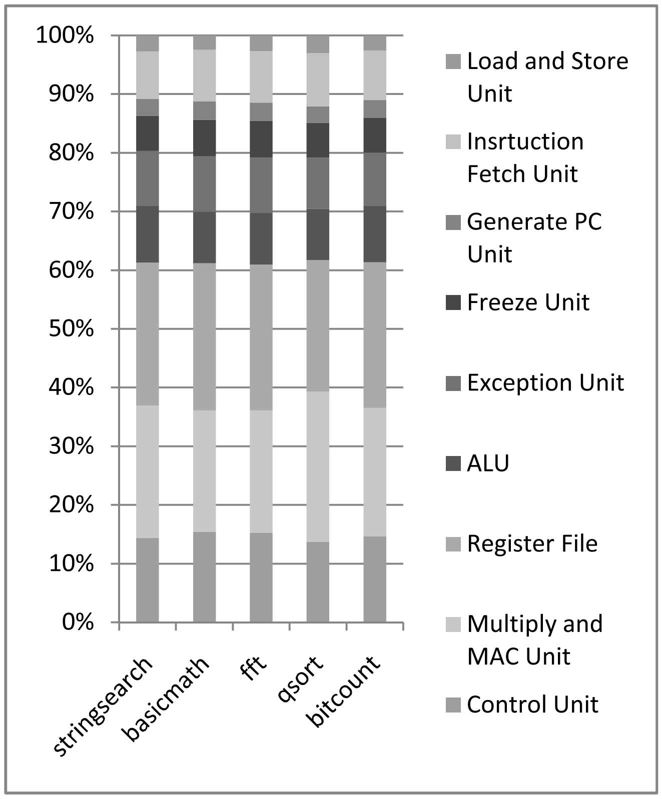

Figure 5-Breakdown of Power Consumption among Various CPU Components

bit/loop counter, it actually consumes 1.14 times more energy.

Second, these results clearly show the importance of choosing algorithms in the process of designing embedded software. Proper algorithm choice can lead to significant energy savings as well as performance improvements, in this case up to 85% and 89% respectively when comparing the best algorithm with the worst one, as these decisions are made at the highest level of the design process.

5. EFFECT OF COMPILER OPTIMIZATIONS

In this section, we examine the effect of compiler optimizations on performance and energy consumption of embedded software. Estimates were done on *recursive bit count by nibbles* function, from *bitcount* benchmark. While compiled code of other *bitcount* functions are pretty much the same regardless of optimization level, this function is compiled differently with each level of optimization, enabling us to fully investigate the effect of each level. Power and energy estimates are shown in figures 3 and 4 respectively, for four optimization levels of GCC. *O0* and *Os* indicate the non-optimized version and size-optimized version of the function. Power savings gained with compiler optimizations are largely due to removing useless load and store instructions from the non-optimized version. These instructions use approximately 30% more power than integer instructions. However, considering the big difference between non-optimized and optimized codes in terms of energy, it is apparent that the dominant factor reducing energy consumption is the shorter run-time rather than less-power-consuming instructions.

These observations, along with the fact that improvements in embedded software performance will also reduce switching-independent (leakage) energy which is constantly increasing its share in system total energy consumption with smaller technologies [11], point to two important considerations for designing embedded software:

First, performance optimizations even with slight power penalties should be the top priority. Second, power reduction techniques which target dynamic power should always be double checked in terms of their effect on system's total energy dissipation. In other words, dynamic power reduction techniques which have even slight performance penalties should not be implemented merely because they reduce switching activity more than they limit performance.

6. FURTHER OPTIMIZATIONS

Compiler optimizations targeting power reduction in embedded processors have been mainly focusing on reducing switching activity on the instruction bus by re-scheduling instructions in such a way that minimum

	Op. Code						Destination Reg.					Input Reg.					Immediate															
Inst/Bit#	31	.	.	.	.	26	25	.	.	.	21	20	.	.	.	16	15	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0
l.addi	1	0	0	1	1	1	0	0	0	1	1	0	0	1	0	0	1	1	0	1	0	0	0	1	0	1	1	0	1	1	0	1
l.ori	1	0	1	0	1	0	0	1	0	1	0	1	0	0	1	0	1	0	0	1	0	0	0	1	0	0	1	0	1	1	0	1
l.addi	1	0	0	1	1	1	0	1	1	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	1	0	0	1	0	0	1	0

Figure 6-Re-sheduling Example. Considering same op. code and input register, second *l.addi* should replace *l.ori.*

switching happens when the processor fetches the next instruction. These techniques, however, neglect the importance of hardware-dependency of compilers. In this section, we investigate other compilation techniques with an eye on the underlying hardware. To this end, we estimated power consumption of various CPU components while a number of benchmarks were executed on the processor. The results are shown in figure 5.

As can be seen in this figure, more than 60% of CPU power is consumed in control unit, multiply and MAC unit, and register file. While high power consumption of multiply and MAC unit is mainly because of not considering power consumption in hardware design of OpenRISC and can be easily fixed using hardware techniques (by detaching its inputs from the register file when current instruction is not multiply), high power consumption in control unit and register file relies mainly on the software.

Based on these observations, it is reasonable to pay special attention to specific fields of the instructions while re-scheduling. We elaborate on this with a simple example: consider a situation when the compiler is deciding for the instruction to be placed after the first *l.addi* instruction in figure 6. With common scheduling techniques, *l.*ori instruction will be chosen by the compiler as it imposes less switching on the instruction bus. Specifically, *l.ori* instruction switches 10 lines of the instruction bus, while the second *l.addi* instruction switches 19 lines. However, if we pay special attention to the specific fields of the instruction, we may find out that the second *l.addi* instruction consumes less power as it imposes no switching on the inputs of the control unit (since there is no change in the op code) and one of the register file ports.

Another optimization technique which can be applied, considering the specific properties of the underlying hardware, is dynamic usage of reserved bits in the binary representation of the instructions. Many instructions of the embedded processors include several reserved bits in their binary representation. . In OpenRISC 1000 instruction set, for instance, more than 15% of the bits of an instruction are reserved on average. This provides a considerable potential opportunity to reduce dynamic power consumption. These bits can be used dynamically, after all other optimizations and scheduling, to reduce switching activity on the instruction bus.

7. CONCLUSION AND FUTURE WORK

We studied the effects of a number of embedded software optimizations on performance and energy consumption of OpenRISC processor core. Our experiments showed that algorithm choice can significantly improve performance and energy consumption. Compiler optimizations also showed large improvements whereas different levels of optimizations did not considerably change performance or energy consumption. After identifying power-hungry components of the CPU, we proposed energy reduction techniques at compiler level. These techniques, unlike previous ones, have an eye on the underlying hardware when scheduling and binary conversion of the compiled code.

As our future work, we will implement the proposed techniques in a tool to further optimize the compiled code of GCC. We will measure their effect quantitatively. Also, further software and hardware power optimizations will be studied.

8. REFERENCES

[1] M. Pedram, "Power optimization and management in embedded systems," *Proceedings of the ASP-DAC Design Automation Conference 2001, Asia and South Pacific*, pp. 239-244, 2001.

[2] T. Simunic, G. de Micheli, L. Benini, and M. Hans, "Source code optimization and profiling of energy consumption in embedded systems," *International Symposium on System Synthesis*, pp. 193-199, Sept. 2000.

[3] V. Dalal and C.P. Ravikumar, "Software power optimizations in an embedded system," *Fourteenth International Conference on VLSI Design*, pp. 254-259, Jan. 2001.

[4] D.A. Ortiz and N.G. Santiago, "High-level optimization for low power consumption on microprocessor-based systems," *50th Midwest Symposium on Circuits and Systems,* pp.1265-1268, Aug. 2007.

[5] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol.2, no.4, pp.437-445, Dec. 1994.
1
[6] S. Watanabe, T. Sato, "Uncriticality-Directed Low-Power Instruction Scheduling," *IEEE Computer Society Annual Symposium on VLSI*, pp.69-74, April 2008.

[7] Chen Yong, He Yanxiang, Liao Ximi, Wu Wei, Li Qingan, "Dynamic probability based instruction scheduling for low-power embedded system," *International Conference on Computer Application and System Modeling,* pp.V2-15-V2-19, Oct. 2010.

[8] M.M. Mansour, I. Hajj, N. Shanbhag, "Instruction scheduling for low power on dynamically variable voltage processors," *The 7th IEEE International Conference on Electronics, Circuits and System*, pp.613-618 vol.1, 2000.

[9] OpenRISC 1200 Specification, *OPENCORES.ORG*, http://opencores.org/openrisc,or1200.

[10] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, R.B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," *Workload Characterization*, pp. 3- 14, 2001.

[11] A. Sinha, A.P. Chandrakasan, "Energy aware software," *Thirteenth International Conference on VLSI Design,* pp. 50-55, 2000.