EProf: An Energy Profiler for the iPAQ

by

Kelly Koskelin

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2004

Author .....
Department of Electrical Engineering and Computer Science
February 4, 2004

Certified by .....
Krste Asanovic
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

EProf: An Energy Profiler for the iPAQ

by

Kelly Koskelin

Submitted to the Department of Electrical Engineering and Computer Science
on February 4, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

In this thesis, I designed and built EProf, a system that profiles the energy use of a Compaq iPAQ. Energy profilers help determine what parts of code are most energy-intensive so that programmers can concentrate on software hotspots. EProf uses statistical sampling to measure an iPAQ's energy use under a variety of working conditions. The EProf infrastructure is a foundation for further work on portable, online energy profiling.

Thesis Supervisor: Krste Asanovic
Title: Associate Professor

Acknowledgments

I would first like to thank my advisor, Krste Asanovic, for giving me the opportunity to work on this project. His guidance and immense patience and understanding were crucial for my success. I would also like to thank the members of the Assam group for their support. Ken Barr was especially knowledgeable and accessible throughout this process. Ken Steele was a great iPAQ hardware resource.

Many thanks to Yaoyao and Taka for their delicious meals and encouragement. Lars and the Bat Boy helped more than they will ever know.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Efforts to extend battery life have become more crucial as the use of portable electronic devices has become increasingly widespread. Much work has gone toward improving the efficiency of hardware components and recently more software techniques are being developed to help conserve energy. In the past when more emphasis was placed on performance, cycle profilers gave programmers an idea of where the processor was spending its time [8]. Now that extra consideration is placed on energy-efficiency, profilers are being developed to correlate energy use with functions or instructions within a program. These energy profilers allow programmers to see where most energy is consumed, enabling them to write more energy-efficient code.

This thesis describes EProf, a collection of hardware and software tools that are used to profile the energy use of the Compaq iPAQ [1]. EProf uses statistical sampling techniques to measure the iPAQ's energy consumption, and correlates these samples with the appropriate process running in the system. EProf is a platform to further develop techniques for measuring energy outside of a laboratory environment. With appropriate packaging, EProf can profile the iPAQ's energy use without being tethered to laboratory equipment because all the energy measurement hardware fits on a PCMCIA Card. EProf can measure the energy consumed under typical working conditions, even as factors such as battery level vary. It is also a flexible tool because the only hardware modification necessary is the addition of a sense resistor to the iPAQ's battery pack.

1.1 System Overview

The use of statistical sampling techniques for energy profiling borrows from work done on cycle profilers [8]. A cycle profiling system periodically interrupts the processor in order to record the executing instruction. The system keeps track of how many times each instruction is interrupted, thus building a statistical profile of where the processor spends its time.

EProf periodically samples the program counter and process identifier just as is done in a cycle profiler. Additionally, it records information to determine the amount of energy used since the last interrupt. The amount of energy used is computed by integrating power samples that are continuously taken in the background. The instantaneous power samples are computed by sampling the current and voltage being supplied to the iPAQ. A separate processing system reports the amount of energy used by each process as well as supplementary data such as average per-process instantaneous power and current.

1.2 Paper Overview

This thesis is structured as follows. Chapter 2 reviews related energy measurement work. Chapter 3 addresses issues related to the design and implementation of the EProf system. Chapter 4 presents experiments and testing. Chapter 5 concludes the thesis and suggests directions for future work.

Chapter 2

Related Work

This chapter addresses two classes of relevant energy measurement techniques. There are a few closely related statistical sampling systems for in-laboratory use. Also, some non-statistical sampling based work supports the efficacy of EProf's energy measurement techniques.

2.1 Statistical Sampling

Basic cycle profilers give the user an idea of where the processor spends its time. However, because all the components of a system, not just the processor, consume energy this does not correlate well with overall energy use. Two types of statistical sampling techniques have been developed for energy measurement; time-driven and energy-driven.

2.1.1 Time-driven Sampling

PowerScope [9] is an example of a time-driven statistical sampling system. A laboratory multimeter periodically interrupts the system to signal that it is time to sample the program counter, process identifier, and instantaneous current being drawn by the system. This is enough information to determine the instantaneous power drain because the system is powered by a lab supply. Because the battery is unplugged the

variation in supply voltage is minimal, thus PowerScope energy profiles do not take into account the effects of battery charging and discharging. Also, since the system relies on the multimeter and lab supply, it cannot be used outside of lab.

2.1.2 Energy-driven Sampling

Energy-driven sampling techniques determine when to take a sample based upon the amount of energy consumed rather than the amount of time that has passed [6]. The profiler issues an interrupt when a given quanta of energy has been used. The interrupted instruction is then recorded and time-stamped. One advantage of the energy-driven approach is that when a lot of energy is being used, a series of interrupts will be triggered in quick succession. This allows analysis tools to pinpoint the expensive processes and catch energy peaks that might be missed in the time-driven scheme. Also, when the system is in an idle state using little power, fewer samples will be taken, saving storage space and reducing the amount by which the system is disturbed. In F. Chang et al.'s energy-driven sampling paper, the authors claim that their implementation of a PowerScope-like profiler tends to over-estimate kernel idle power [6]. EProf samples power continuously so the associated power sampling overhead is constant. Much like the energy-driven scheme, most consideration needs to be placed upon disturbances caused by the software instruction sampler.

2.2 Energy characteristics of embedded processors

There are a number of approaches to energy measurement that do not involve statistical sampling which offer results that are relevant to EProf's techniques.

The JouleTrack [11] system illustrates that processor speed and voltage are the two factors that most affect the amount of energy used by the system. EProf takes voltage supply variation into account whereas the above in-lab systems ignore it.

Isci and Martonosi [7] are critical of time-based statistical sampling when applied to complex processors with multiple power states. Similarly, F. Chang et al. [6] report that energy-driven estimates are more accurate when the processor cycles

through a number of different power states. The iPAQ's processor has only two power states, but the system as a whole is much more complex. One must consider separate components such as DRAM, screen, and wireless card; a wireless card alone has a number of different power states. Future work should address these considerations, but even so, most measurements by the energy-driven sampler and PowerScope were within 4 percent of eachother [4].

Chapter 3

Design and Implementation

This chapter addresses the architecture of the EProf system. It describes the energy measurement strategy as well as design and implementation details.

3.1 Design Overview

EProf is composed of a hardware subsystem that measures instantaneous power and a software portion for sample recording and data analysis. It is possible to set the rates of power sampling and instruction sampling independently, however, the hardware system samples power at least as often as instructions are sampled. Figure 3-1 illustrates how power samples are integrated and energy is charged to the interrupted instruction. The alternating shaded regions are the groups of power samples attributed to the next interrupted instruction, indicated by a heavy black line.

Consider two sequential interrupts in two separate processes. The second process is charged for any energy used in between interrupts regardless of where the majority of the intervening time slice is spent. One advantage of this technique is that many power samples can be taken by the EProf card while minimally disturbing the rest of the system, resulting in a finer-grained analysis. However, this is only useful if the power samples are charged to the appropriate process. The problem of properly attributing power samples makes it crucial to look at the way the instruction and energy sampling rates interact.

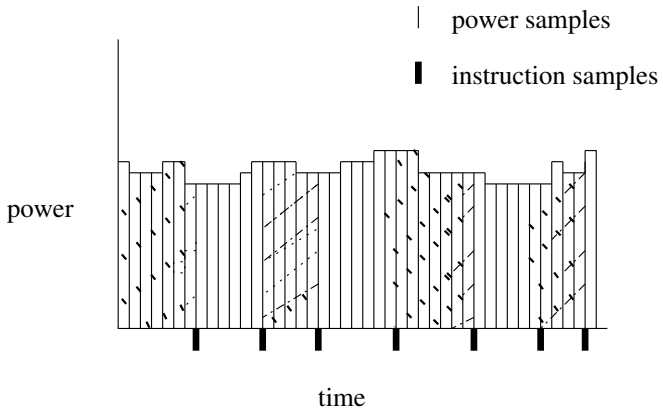


Figure 3-1: Energy allocation. The instruction sampling rate is jittered to avoid synchronization with system events.

3.2 Implementation Overview

EProf was developed on a Compaq iPAQ H3600 [1] running the Familiar distribution of Linux [2]. The H3600 contains a 206 Megahertz (MHz) Intel StrongArm SA1110 processor and plugs into an iPAQ backPAQ sleeve with two PCMCIA slots. One slot houses the EProf PC Card, the other can be used by any other PC Card. In this application the second slot enables the profiling of programs that use the wireless card.

3.2.1 Hardware

Most of the power-measurement portion of EProf sits on a card that plugs into a PCMCIA slot on the iPAQ's backPAQ. The EProf card consists of an oscillator, two analog to digital converters (ADCs), and a Field Programmable Gate Array (FPGA).

Overview

One of the ADCs measures the voltage drop across a sense resistor that is placed between the battery and the battery connector to the iPAQ. This allows EProf to determine the current being drawn by the system. The other ADC measures the voltage being supplied by the iPAQ's battery. This is necessary because the supplied

voltage will vary as the battery charges and discharges. Because instantaneous power is equal to current times voltage, the above information is sufficient to determine the power being used at the time of the sample. The FPGA controls the ADCs and passes their samples to the iPAQ via the PCMCIA port for processing.

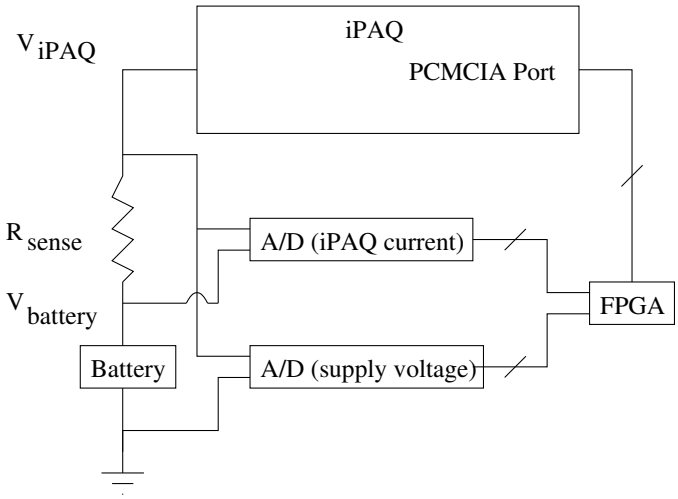


Figure 3-2: EProf Hardware System Overview.

Implementation

The 20 MHz oscillator clocks the state machines inside the FPGA. The FPGA generates the 588 Kilohertz (kHz) clock signal used by the ADCs as well as the ADC chip enable signals that allow sample timing to be finely controlled by the user. The ADCs are 8-bit, have differential inputs, and use successive approximation. The sense resistor is 0.5 Ohm. The battery is rated to provide a maximum of 1 Amp of current so this introduces a drop of at most 0.5 Volt over the resistor. The FPGA is a 3.3 Volt Xilinx SpartanXL which easily interfaces with the data and control signals needed for the PCMCIA slot. The FPGA contains one state machine to control sampling and one to respond to user input.

The sampling state machine generates the ADC clock and control signals and aggregates sample data. At the end of a conversion the resulting 8-bit sample is added to the corresponding voltage aggregate buffer and a sample counter is incremented.

When the user reads sample data the two 16-bit totals are returned, along with the counter data indicating how many voltage samples the aggregates correspond to. The registers can hold 255 samples before overflowing, so the user must be careful to coordinate the instruction and energy sampling rates to avoid this situation. Also, all the buffers are cleared when the sample counter register is read, so all three buffers must be read sequentially to ensure that the voltage aggregates correspond to the number of samples returned by the counter.

Name	*Size*	*Read/Write?*	*Description*
reset	8-bit	W-Only	write any value to reset card
sample_rate	16-bit	R/W	power is sampled at a rate of $adc\_clk \div (10 + (sample\_rate\_reg - 1))$
adc0_sample	16-bit	R-Only	aggregate supply voltage
adc1_sample	16-bit	R-Only	aggregate voltage drop across sense resistor
sample_counter	16-bit	R-Only	number of samples above aggregates correspond to

Table 3.1: EProf card registers.

The iPAQ's interface to the PC Card slot/EProf driver is via the second state machine in the FPGA. This machine handles reads and writes to the EProf's five registers, summarized in Table 3.1. The three read-only data registers were described above; there are two 16-bit aggregate voltage sample buffers and a 16-bit sample counter. Writes to the reset register clear all buffers and return the card to its starting state. The 16-bit sample_rate register is read/write and determines the ADC sampling rate. The ADC clock rate is given by $adc\_clk = (20 * 10^6) \div 34 \approx 588$ kHz; it must be a multiple of the 20 MHz clock, but less than 600 kHz. Because sample conversion takes 10 ADC clock cycles, the maximum power sampling rate is given by $adc\_clk \div (10 + (sample\_rate\_reg))$, as long as sample_rate_reg is nonzero. The fastest power sampling rate of approximately 49 kHz is achieved when the sample counter is one.

The ADCs are 8-bit Texas Instruments TLC0831s that use 5 Volt control signals. The TLC0831 is a successive approximation analog to digital converter with a ref-

erence voltage input that allows for the accurate measurement of signals less than 5 Volts. Its differential inputs enable accurate voltage measurements even when the negative terminal is not connected to ground. However, the differential inputs are directional so the voltage drop across the sense resistor can only be measured when charge is flowing out of the battery. When a negative voltage is detected, the ADCs return a sample value of 0. This means that EProf cannot be used when the battery is charging. One solution to this problem would be to disallow charging of the iPAQ's battery when it is plugged into the sleeve and the sleeve is plugged into the wall outlet. However, this would involve nontrivial modifications to the sleeve to disable the iPAQ's battery charge enable pin.

3.2.2 Software

EProf's software subsystem is comprised of online tools needed for profiling and offline tools for data analysis and debugging. The online tools consist of two loadable kernel modules and a user-level daemon, eprofd, to control sampling. Additionally, there is a command-line interface to the EProf card and perl and shell scripts for offline data analysis. An overview of the organization is shown in Figure 3-3.

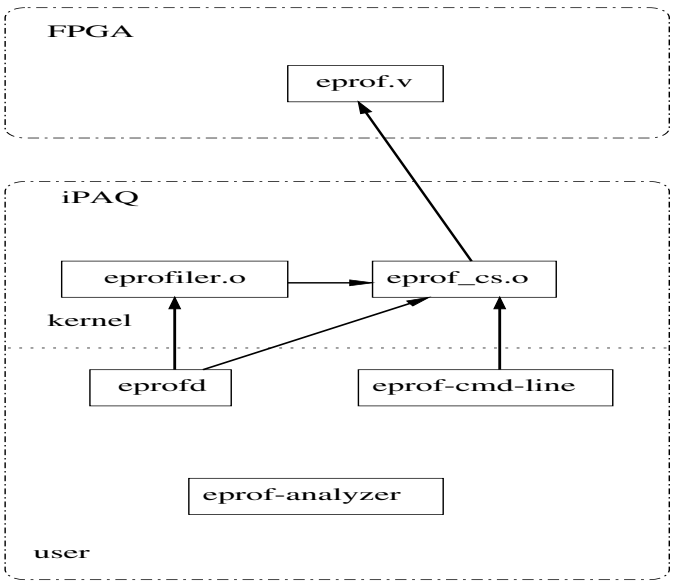


Figure 3-3: EProf Software System Overview. Arrows indicate dependencies.

Kernel code

The kernel level code is split into two modules. The EProf card driver exports a character device interface that allows programs to read and write the card's control registers. It also exports the eprof_read function used by the profiling module to read energy sample data. The profiling module exports a separate character device interface that the user-level daemon uses to read the instruction and energy samples.

The EProf card driver is based on a basic memory card driver that is part of the Card Services for Linux PCMCIA support package (pcmcia-cs) available from SourceForge [3]. Reads and writes to the card directly manipulate the registers summarized in Table 3.1. The exported eprof_read function allows the profiling module to read energy samples.

The profiling module is a slightly modified version of the cycle profiler distributed with Familiar. During the interrupt that records instruction information, the profiling module reads the latest energy samples from the EProf card. It provides the API that the user-level daemon uses to start profiling, stop profiling, and control instruction sampling parameters.

To reduce overhead, a future implementation of EProf might more tightly integrate the two modules. For example, rather than exporting the eprof_read function, the two modules might share a kernel memory buffer. It might also be advantageous to combine them; the functionality might be separated by creating two devices with different minor numbers that are serviced by one combined EProf module.

User-level sampling control

The user-level daemon is invoked at the command-line and allows the user to specify both the instruction sampling rate and the energy sampling rate. The default instruction sampling rate is 625 Hz which is the rate used by PowerScope [9]. The default energy sampling rate is 49 kHz which is the fastest sampling rate achievable by the hardware. The user may also randomly jitter the sampling rate around a specific value. This helps guard against samples that are synchronized with system events.

Analysis and debugging tools

The command-line interface to the EProf card allows the user to directly manipulate the card's registers. From the command line the user the can start or stop sampling, change the power sampling rate, or read the most recent samples. Currently the EProf card driver does not distinguish between reads from the profiling module and the command-line program. If the eprofd daemon is running the user must be careful not to clear the sample buffer or this will result in eprofd reading an incomplete power sample.

The main offline analysis tool is a perl script that can be run on any machine. It displays average power/instruction in a given process, average instantaneous current and voltage, converts the power samples to energy values and correlates them with the appropriate process. The tool also displays the incremental percentage of total energy consumed. Example output is shown in Table 3.2.

cycles	*cycle %*	*cum cycle%*	*image*
68731	71.23%	71.23%	kernel
27598	28.60%	99.83%	loop
136	0.14%	99.98%	profd
13	0.01%	99.99%	ld-2.2.3.so
8	0.01%	100.00%	libc-2.2.3.so
2	0.00%	100.00%	sh
1	0.00%	100.00%	init

energy (Joules)	*energy %*	*cum energy %*	*image*
106.79330	66.58%	66.58%	kernel
53.26959	33.21%	99.79%	loop
0.28268	0.18%	99.97%	profd
0.02627	0.02%	99.99%	ld-2.2.3.so
0.01708	0.01%	100.00%	libc-2.2.3.so
0.00389	0.00%	100.00%	sh
0.00192	0.00%	100.00%	init

# power samples	*power/samp*	*avg V*	*avg I*	*avg e/i*	*image*
5270376	0.97112W	3.703V	0.262A	76.7	kernel
2123784	1.20637W	3.682V	0.328A	77.0	loop
10633	1.29906W	3.645V	0.356A	78.2	profd
996	1.26283W	3.652V	0.346A	76.6	ld-2.2.3.so
631	1.33460W	3.653V	0.365A	78.9	libc-2.2.3.so
154	1.21541W	3.603V	0.337A	77.0	sh
80	1.19804W	3.696V	0.324A	80.0	init

Table 3.2: Example energy profile.

Chapter 4

Experimentation and Results

This chapter presents some of the data that has been collected using the EProf system. It addresses the effects of varying the instruction sampling rate versus varying the energy sampling rate. This chapter also compares programs with differing energy characteristics and evaluates a profile of the iPAQ's energy consumption as its battery drains.

4.1 Testing Setup

For all testing the wall charger was unplugged so the batteries of both the sleeve and iPAQ were being drained. The ADCs and clock were powered by the 5 Volt pins from a PCMCIA slot on the backPAQ. The ADC measuring the voltage supplied by the 3.7 Volt iPAQ battery also used one of the slot's power pins as its reference voltage. The other ADC used the lab power supply to get the 1 Volt reference needed to measure the voltage drop across the sense resistor. (See 3.2.1 for more information.) Instruction sample rate jittering was enabled to minimize correlating instruction samples with system events. The looping program simply incremented a counter from 0 until 0x2fffffff. This lasted an average of 46 seconds.

4.2 Varying the Sampling Rates

Energy measurements were taken using six different energy sampling rates and six different instruction sampling rates. When the energy sampling rate was varied the instruction sampling rate was kept at a constant 625 Hz [9]. When the instruction sampling rate was varied energy was sampled at the highest possible rate, 49kHz. During testing the profiling daemon was run in the background while a single instance of the looping program was run in the foreground. Each test was repeated three times and the figures show the the average of the three test runs.

Figure 4-1 shows the average instantaneous current being drawn when running kernel code, the loop code, and other user-mode code. Although there is some noise in the system, the variation in current is within a few percent. Also, both plots show the current drain increasing as sampling frequency increases; one would expect to incur higher overhead when taking more samples. The tests also show more current being drawn by kernel code than by the loop code. The loop code is CPU-bound so the rest of the system is relatively quiescent when it is running. The kernel is likely to be using other parts of the system, thus drawing more current from the battery.

Figure 4-2 shows the average voltage sampled for the different processes running in the system. The measured voltage is strongly correlated with the time that the instruction was sampled. This explains why the graphs in Figure 4-3 showing average instantaneous power have a shape similar shape to those in Figure 4-1.

The tests varying the energy sampling rate were run in order from highest frequency to lowest frequency so the supply voltage seems to increase with frequency. The tests varying instruction sampling rate were run in order from lowest frequency to highest frequency so the voltage seems to decrease as frequency increases. The effect of time on battery voltage is illustrated further in Section 4.4.

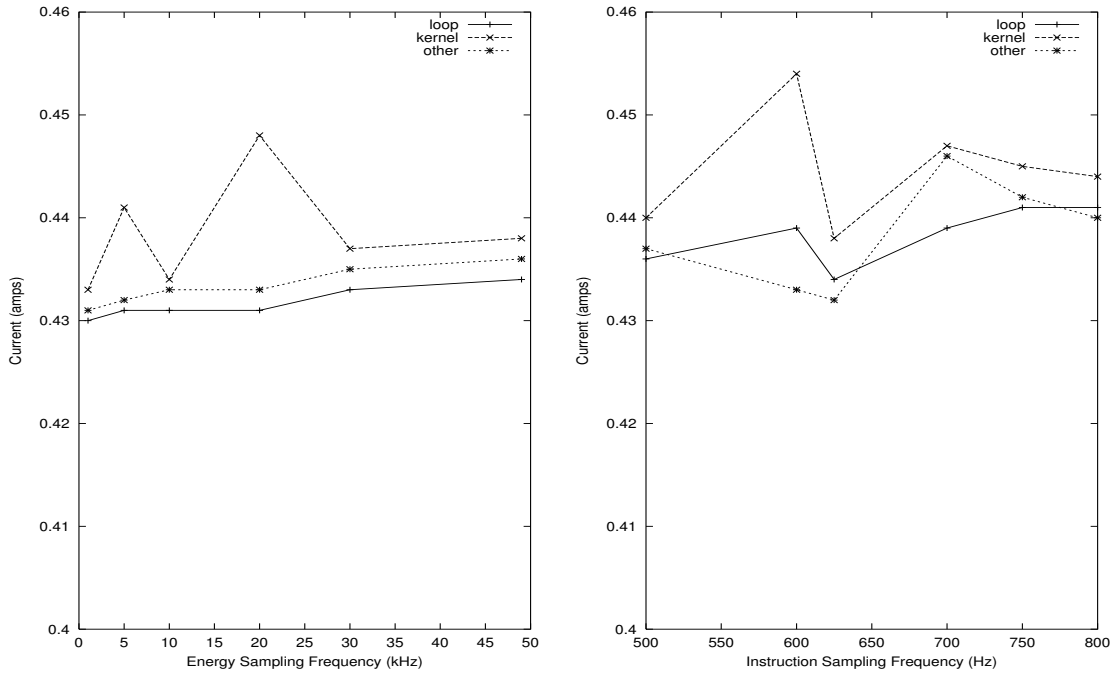


Figure 4-1: Average instantaneous current samples changing with energy and instruction sampling rate.

4.3 Different Energy Patterns

In order to observe differences in the energy characteristics of distinct processes, two different looping programs were profiled simultaneously. With the profiling daemon running in the background, a modified loop-writing program synchronously wrote 4 bytes to flash memory then incremented a counter. After 0x3ff iterations of the loop-writing program, the original loop program was run. Whereas the unmodified program took 46 seconds to run on average, the modified program lasted between 25 and 100 seconds depending on the timing of the flash memory accesses [1].

Figure 4-4 shows the resulting effects on average instantaneous current, voltage, and power for three runs of the test described above. These averages are shown for only the most commonly interrupted processes. Notice that the modified loop-writing program does not appear in the list, and that the kernel uses around 1 milliamp less current on average than other processes. Manual inspection of the detailed results returned by the profiling daemon reveal that the kernel is spending most of its time

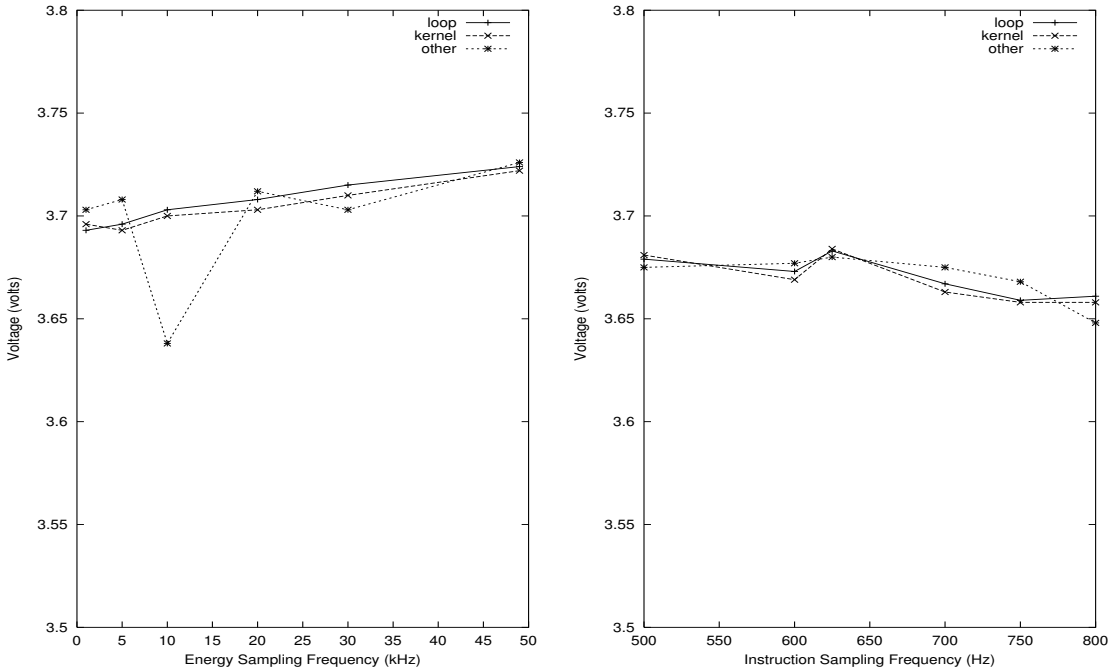


Figure 4-2: Average instantaneous voltage samples changing with energy and instruction sampling rate.

waiting for the synchronous writes to complete. The time spent waiting for the flash memory lowers the kernel's average current drain. Figure 4-4(b) shows an inverse relationship between supply voltage and current, while Figure 4-4(c) shows that power is strongly correlated with current drain.

Although Figure 4-4 shows a high variance in instantaneous power, Table 4.3 demonstrates consistency in total energy use. The original loop program uses an average of 53 joules, though the percentage of overall energy consumption varies depending on how long the test lasts. Interestingly, regardless of test length the average amount of energy attributed to each power sample (shown in column 6) varies little. This emphasizes that although minimizing current draw may be helpful, the amount of time that a program runs is still the most important factor in determining energy consumption.

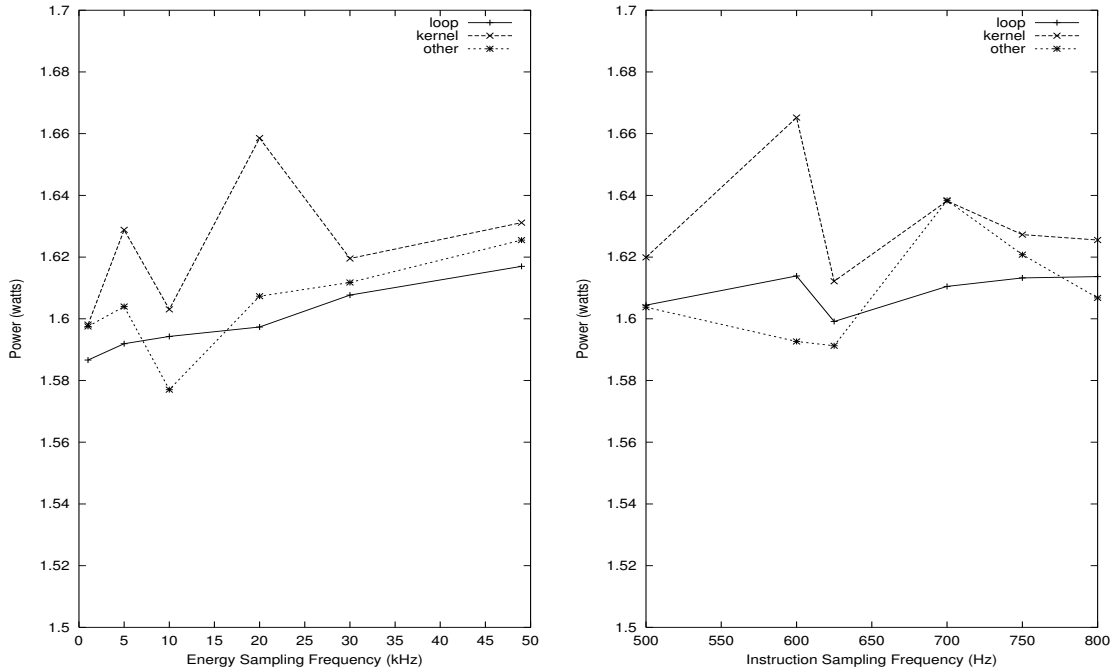


Figure 4-3: Average instantaneous power samples changing with energy and instruction sampling rate.

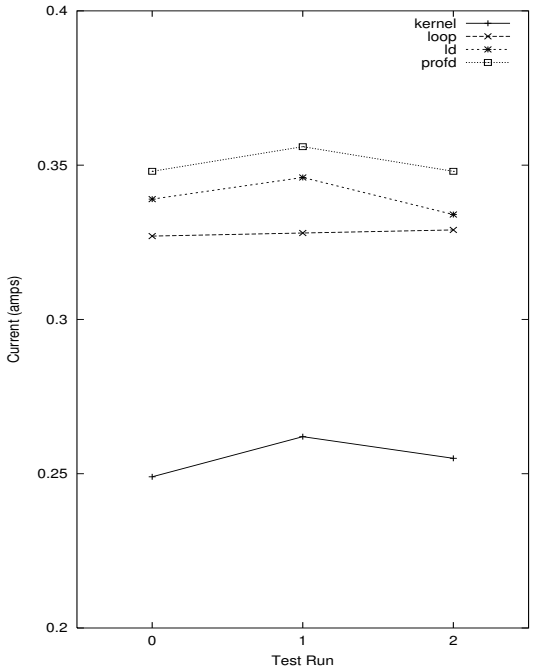
4.4 Time Varying

Energy measurements were taken as a full battery was draining in order to investigate the way that battery level affects current and energy use. A script on the iPAQ continuously ran iterations of the loop program and slept for one second in between invocations. Figure 4-5(a) shows the average instantaneous current drawn by the loop program over time, while figure 4-5(b) shows average instantaneous voltage. When the battery voltage is high, the amount of current drawn is inversely proportional to voltage. After about 15 iterations of the loop program, when the battery voltage drops below 3.6 Volts, the current operates in an almost linear region between 0.46 Amps and 0.48 Amps. However, once the voltage gets below 3.4 Volts the amount of current drawn increases dramatically. Figure 4-5(c) shows the resulting effect on average instantaneous power. As long as current stays constant, power decreases. As soon as voltage drops below the critical level the current and power increase quickly. Figure 4-5(d) shows that despite the fluctuations in average instantaneous power total

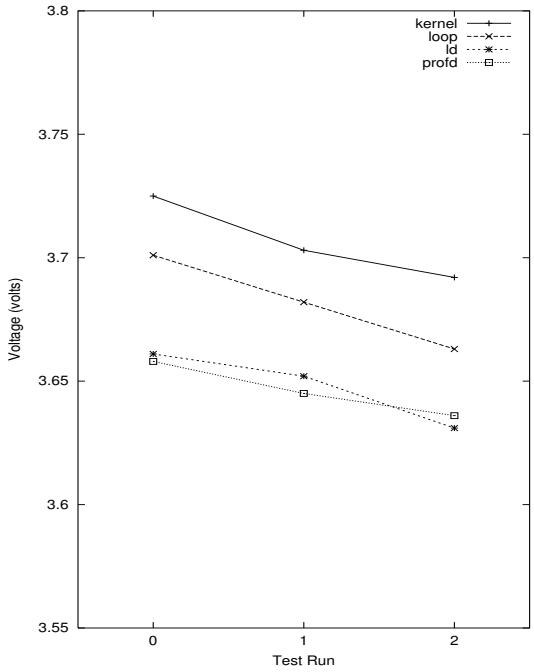
Test	*Loop Energy*	*Total Energy*	*% of Total*	*# Power Samples*	*Energy/Sample*
0	52.79 (J)	89.49 (J)	59 %	4002377	$2.24e^{-}5$
1	53.27 (J)	160.39 (J)	33 %	7406654	$2.17e^{-}5$
2	54.77 (J)	123.52 (J)	44 %	5684369	$2.17e^{-}5$

Table 4.1: Summary of energy characteristics of different programs. All energy is expressed in joules (J).

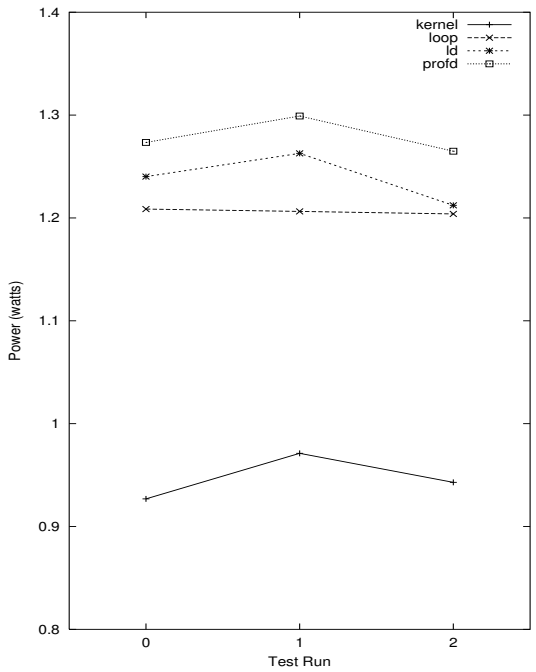
energy consumption remained relatively constant. The unusual data points are likely due to characteristics of the converter used to regulate the sleeve's voltage.



(a) Current

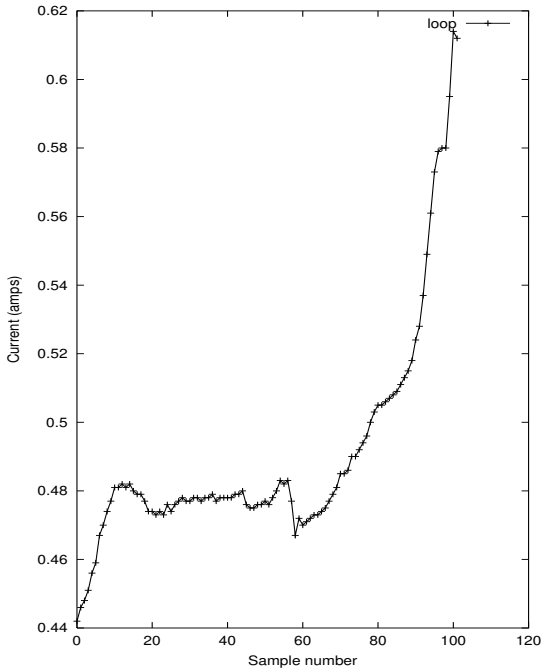


(b) Voltage

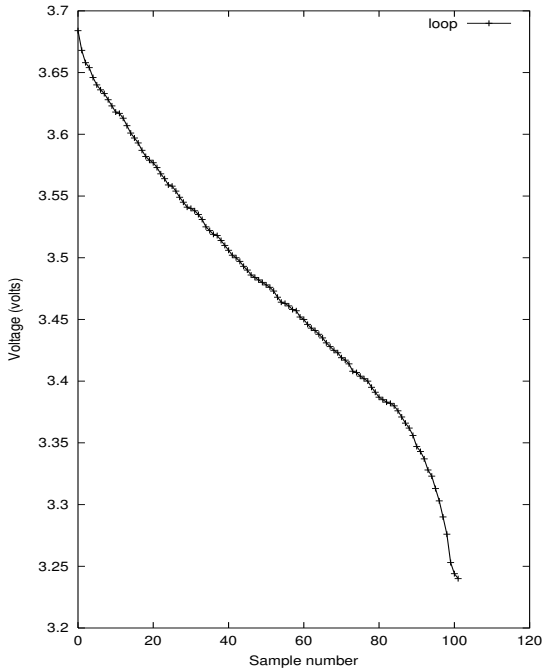


(c) Power

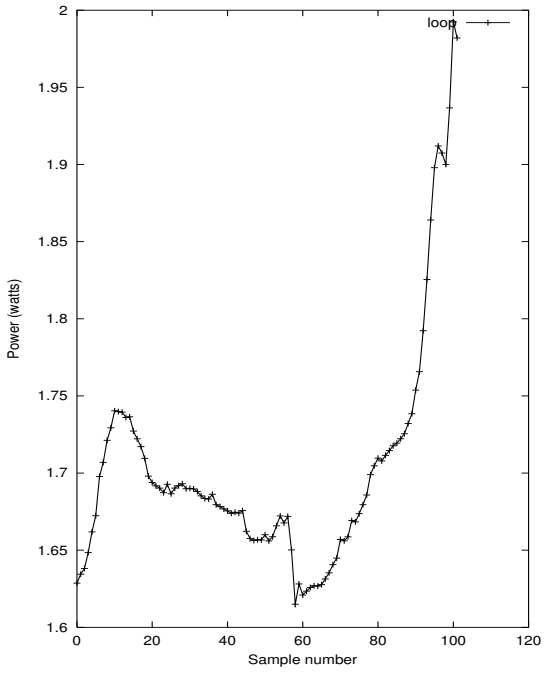
Figure 4-4: Examples of different energy sample characteristics.



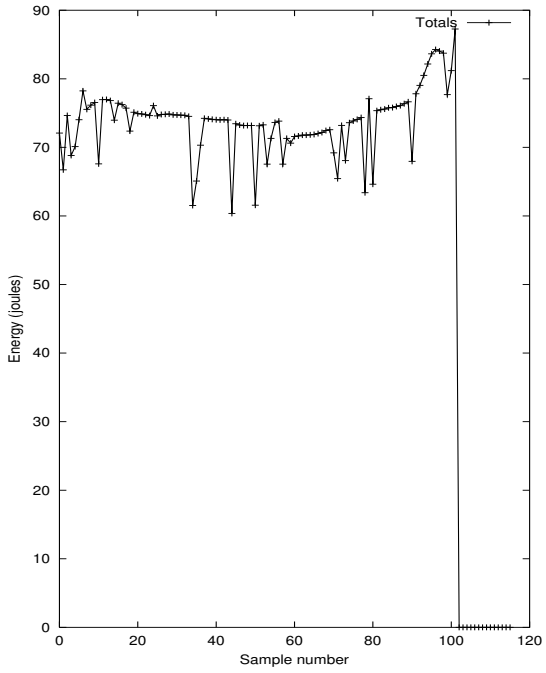
(a) Current



(b) Voltage



(c) Power



(d) Energy

Figure 4-5: Energy samples changing over time.

Chapter 5

Conclusion and future work

This thesis presented the EProf system, an infrastructure for the development of a portable, online energy monitoring tool. It focused on the design of the system, issues involved in achieving its design goals, and a preliminary investigation of EProf's effectiveness as an energy profiler.

Future research on the EProf platform can proceed in a number of directions. For example, one might focus on making EProf more portable. This is a challenging project because low-power components are needed to build an efficient power distribution system on the EProf card. A portable tool is likely to have greater storage needs, so data compression techniques will be useful to keep the amount of space used from becoming unwieldy. It may be valuable to investigate an energy-driven version of EProf to evaluate its accuracy; an energy-driven version may incur less overhead over all. One might add hooks to the operating system to cause samples to be taken upon process switches rather than on time-driven or energy-driven events; it is not clear whether the improvement in accuracy would justify the added complexity.

EProf is a solid base upon which further portable energy monitoring projects can build.

Bibliography

[1] Compaq ipaq h3600 hardware documents. http://handhelds.org/Compaq/iPAQH3600/.

[2] Familiar linux. http://familiar.handhelds.org/.

[3] Linux pcmcia card services. http:/pcmcia-cs.sourceforge.net/.

[4] K. Barr. Energy aware lossless data compression, master's thesis. http://citeseer.nj.nec.com/barr02energy.html.

[5] K. Barr and K. Asanovic. Energy aware lossless data compression. http://citeseer.nj.nec.com/article/barr03energy.html.

[6] F. Chang, K. Farkas, and P. Ranganathan. Energy-driven statistical profiling: Detecting software hotspots. In *Proceedings of the Workshop on Power-Aware Computer Systems*, February 2002.

[7] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, page 93. IEEE Computer Society, 2003.

[8] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S. Leung, D. Sites M. Vandevoorde, C. Waldspurger, and W. Weihl. Continuous profiling: Where have all the cycles gone. In *Proceedings of the 16th Symposium on Operating Systems Principles*, October 1997.

[9] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, February 1999.

[10] S. Lee, A. Ermedahl, and S. L. Min. An accurate instruction-level energy consumption model for embedded risc processors. In *Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems*, pages 1–10. ACM Press, 2001.

[11] A. Sinha and A. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In *Design Automation Conference*, pages 220–225, 2001.

[12] T. Simunic, L. Benini, and G. De Micheli. Energy-efficient design of battery-powered embedded systems. In *Proceedings of the International Symposium on Low-Power Electronics and Design '98*, June 1998.

[13] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. In *Proceedings of the International Conference on Computer-Aided Design*, November 1994.