JouleTrack - A Web Based Tool for Software Energy Profiling

Amit Sinha
Massachusetts Institute of Technology
Cambridge, MA 02139
Ph: (617) 253-0164
E-mail: sinha@mit.edu

Anantha P. Chandrakasan
Massachusetts Institute of Technology
Cambridge, MA 02139
Ph: (617) 258-7619
E-mail: anantha@mit.edu

ABSTRACT

A software energy estimation methodology is presented that avoids explicit characterization of instruction energy consumption and predicts energy consumption to within 3% accuracy for a set of benchmark programs evaluated on the StrongARM SA-1100 and Hitachi SH-4 microprocessors. The tool, JouleTrack, is available as an online resource and has various estimation levels. It also isolates the switching and leakage components of the energy consumption.

Keywords

Software energy, leakage estimation, instruction energy

1. INTRODUCTION

Estimation of software energy consumption is becoming crucial in embedded applications. Instruction level power estimation tools have been proposed to computed the energy consumption of a given software [1][2][3]. The basic idea is to run each instruction or short sequences of instruction is a loop and measure the current/power consumption. Their primary drawback, however, is that these tools rely on exhaustive characterization of the energy consumption of the entire ISA (Instruction Set Architecture) and inter-instruction effects. The estimation model is compute intensive, requiring a complete trace analysis of the program's instructions and is therefore slow. Our experiments on the StrongARM, SA-1100 [4] and Hitachi SH-4 [5] microprocessors (two very popular low power embedded processor) show that the variation in the current consumption is quite small. A lot of overheads are common across instructions and as a result the overall current consumption of a program is a weak function of the actual instruction stream and to a first order depends only on the operating frequency and voltage. Second order variations do exist but were measured to be less than 7% for a set of benchmark programs. Therefore, a complete instruction level trace analysis is unnecessary and a simple cycle accurate simulation can be used. We propose a simple fast technique to estimate software energy and estimate second order variations. Initial experiments indicate an accuracy within 3% of actual measurements.

With increasing trends towards low power design, supply voltages are constantly being lowered as an effective way to reduce power consumption. However, to satisfy the ever demanding performance requirements, the threshold voltage is also scaled proportionately to provide sufficient current drive and reduce the propagation delay. As the threshold voltage is lowered, the subthreshold leakage current becomes increasingly dominant. We also outline a methodology to separate the leakage current from the switching current. The leakage current model has less than 6% error for the entire working range of the StrongARM.

DAC 2001,June 18-22,2001, Las Vegas, Nevada, USA.

2. INSTRUCTION ENERGY PROFILING

The experimental setup consisted of the Brutus SA-1100 Design Verification Platform which is essentially the StrongARM SA-1100 microprocessor connected to a PC using a serial link. It can operate from 59 MHz to 206 MHz, with a corresponding core supply voltage of 0.8 V to 1.5 V. The power supply to the StrongARM core was provided externally through a variable voltage sourcemeter with the I/O pads running at a fixed supply voltage. The ARM Project Manager (APM) was used to debug, compile and execute software on the StrongARM. Current measurements were performed using the sourcemeter built into the variable power supply. The instruction and data caches were enabled before the programs were executed. To measure the current that is drawn by a subroutine, the subroutine was placed inside a loop with multiple iterations till a stable value of current was measured.

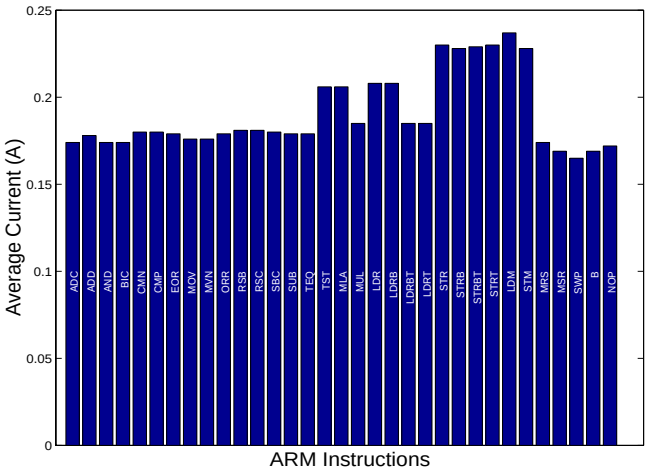

Figure 1. Strong SA-1100 instruction set current consumption

Figure 1 shows the current consumption of all the instructions of the ARM instruction set on SA-1100. Each of the 33 current values are themselves the averages of the various addressing modes and inputs in which the instruction can be executed accounting for a total of about 280 data points. The important point to observe is that the current consumptions are pretty uniform. On an average, arithmetic and logical instructions consume 0.178A, multiplies 0.196A, loads 0.196A, stores

0.229A while the other instructions consume about 0.170A. The total variation in current consumption is 0.072A which is 38% of the overall average current consumption. Figure 2 shows the core current consumption for the Hitachi SH-4 processor running at 2.0V core power supply. The average instruction current is 0.29A, with a variation of 0.11A, which once again is 38% of the average.

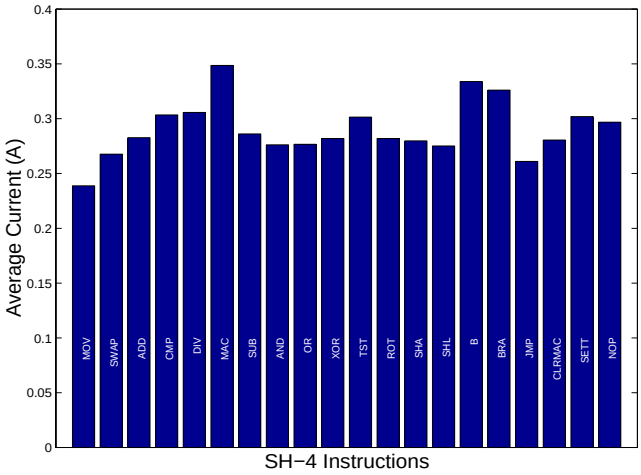

Figure 2. Hitachi SH-4 instruction set current consumption

The current variation within an instruction (as a function of addressing modes and data) is even smaller. Figure 3 shows the current consumption of 3 different instructions as a function of various addressing modes and data. In general, we observed that to a first order the instruction current consumptions are independent of the addressing modes or operands.

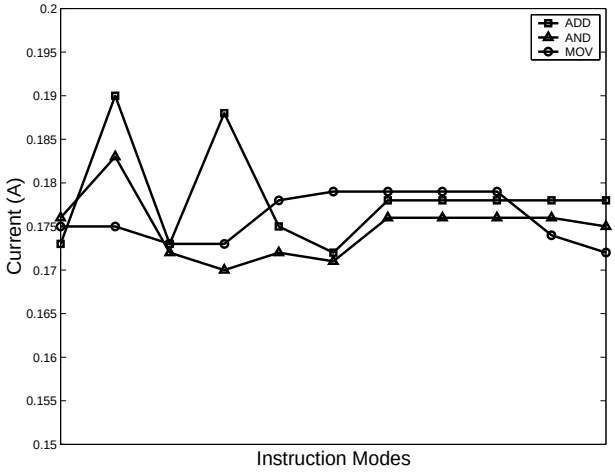

Figure 3. Current variations within instructions

Based on the previous discussion, it is reasonable to conclude that the common overheads (such as caches, decode logic etc.) in contemporary microprocessors are large and overshadow any instruction specific variations. Therefore, estimating software energy consumption with an elaborate instruction trace and inter-instruction analysis is overkill.

3. SOFTWARE ENERGY PROFILING

3.1 First Order Model

While the instruction level current consumption has a variation of about 38%, the variation of the current consumption in programs is much less. Figure 4 shows the current consumption of 6 different benchmark programs at different supply voltage and frequency levels in the StrongARM. The maximum current variation is less than 8%. This implies that to a first order current consumption of a piece of code is independent of the code and depends only on the operating voltage and frequency of the processor. The first order software energy estimation model is then simply

$$E_{tot} = V_{dd} I_0(V_{dd}, f)\Delta t \quad (1)$$

where V_{dd} is the supply voltage and Δt is the program execution time.

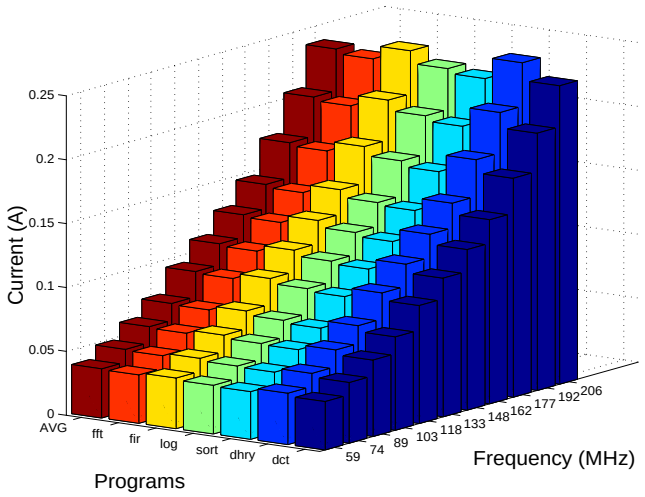

Figure 4. Program current consumption as a function of operating point

3.2 Second Order Model

While the current variation across programs is quite small in the StrongARM, it might be significant in datapath dominated processors. For example, the current consumption of the multiply instruction in DSPs will be far greater than the current consumption of other instructions. In such cases a simple model like Equation 1 will have significant error. The following second order model is proposed.

Let C_k ($k \in [0, K-1]$) denote the K energy differentiated instruction classes in a processor. Energy differentiated instruction classes are partitions of the instruction set such that the average current consumption of any a class is significantly different from that of another class and the current variation within a class is small. Class partitions can also be done on the basis of different cycles, e.g. instruction, data access etc. Let c_k denote the fraction of total cycles in a given program that can be attribute to instructions/cycles in the class C_k, i.e. $\sum c_k = 1$ .The proposed second order current consumption equation is

$$I(V_{dd}, f) = I_0(V_{dd}, f) \sum_{k=0}^{K-1} w_k c_k \quad (2)$$

where w_k are a set of weights. Let $\boldsymbol{W}$ represent the vector $[w_0, w_1, .., w_{K-1}]$. Let P_n ($n \in [0, N-1]$) represent a set of N benchmark programs, $\boldsymbol{C}_n$ denote the cycle fractions vector for program P_n i.e. $[c_0^n, c_1^n, \ldots, c_{K-1}^n]$ and I_n denote its average current consumption. Based on Equation 2, we can express the current vector $\boldsymbol{I}$ in the following form.

$$\boldsymbol{I} = I_0(V_{dd}, f)\boldsymbol{C}\boldsymbol{W} \tag{3}$$

where $\boldsymbol{I}$ is the average current $[I_0, I_1, ..., I_{N-1}]$ for the N programs, $\boldsymbol{C}$ is an $N \times K$ matrix with $\mathbf{C}_n$ as the n^{th} row. The weights can be solved for in an least mean square sense using the pseudo-inverse

$$\boldsymbol{W} = \frac{1}{I_0(V_{dd}, f)}(\boldsymbol{C}^T\boldsymbol{C})^{-1}\boldsymbol{C}^T\boldsymbol{I} \tag{4}$$

If the instruction classes are a valid energy differentiated partition, the weighting vector $\boldsymbol{W}$ will reflect the energy differentiation. The maximum current prediction error will also go down considerably.

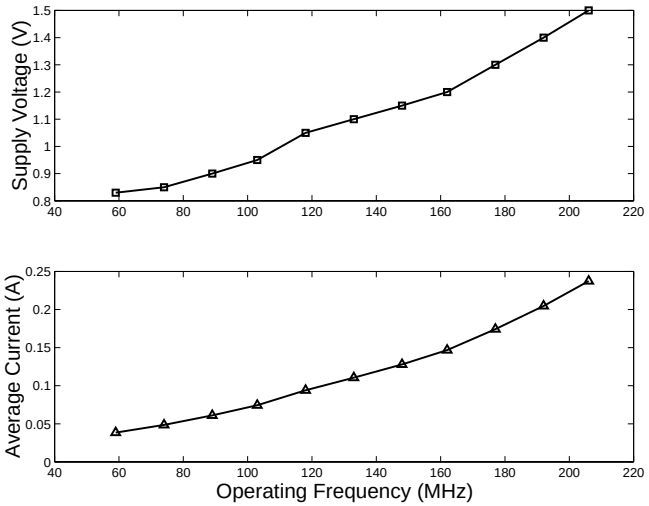

Figure 5. Average current and supply voltage at each operating frequency of the StrongARM SA-1100

On the StrongARM SA-1100, we partitioned the cycles into 4 classes - (i) Instruction, (ii) Sequential memory access (accesses which are related to previous ones) (iii) Non sequential accesses and (iv) Internal cycles. Current measurements were done for 6 benchmark programs running at all possible frequency and voltage combinations. The weighting vector is shown in Table 1. The average current drawn at each operating frequency of the StrongARM is shown in Figure 5. The StrongARM operates at 11 discrete frequency levels. The minimum operating voltage required is also shown and is almost linear with frequency. In fact the normalized operating voltage and frequency are related as

$$\bar{V}_{dd} = 0.66\,\bar{f} + 0.33 \tag{5}$$

where $\bar{V}_{dd}$ and $\bar{f}$ are the normalized to their respective maximum values.

The weighting factors can be interpreted as follows. For programs where instruction cycles and non-sequential memory accesses dominate the current consumption is higher than the average current at that operating point. Internal cycles and sequential memory access dominated programs will have a lower than average current consumption..

TABLE 1 : WEIGHTING FACTORS FOR $K = 4$ ON THE STRONGARM

Class	Weight	Value
Instruction	w_1	2.1739
Sequential memory access	w_2	0.0311
Non-sequential memory access	w_3	1.2366
Internal cycles	w_4	0.8289

The current prediction error with a second order model can be reduced to less than 2%. The advantage is that this accuracy comes for free. No elaborate instruction level profiling is required. Such cycle counts as the ones shown in Table 1 are easily obtained using simulators/debuggers available with standard processors. Figure 6 shows the overall improvement of current prediction accuracy on 66 test points. It can be seen that the current prediction is better in every case (the maximum error of 4.7% using a first order model is reduced to 2.3%). The effectiveness of the current weighting scheme will become more pronounced in processors having a wider variation in average current consumption.

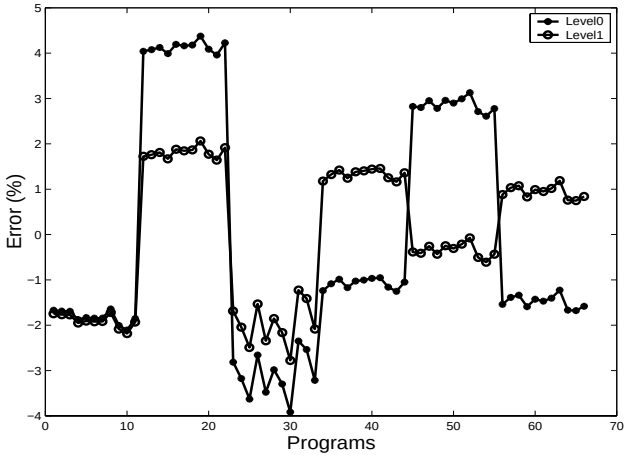

Figure 6. First and second order model prediction errors

4. LEAKAGE ENERGY MODELLING

4.1 Principle

The power consumption of a subroutine executing on a microprocessor can be macroscopically represented as

$$P_{tot} = P_{dyn} + P_{stat} = C_L V_{dd}^2 f + V_{dd} I_{leak} \tag{6}$$

where P_{tot} is the total power which is the sum of the static and dynamic components, C_L is the total average capacitance being switched by the executing program, per clock cycle, and f is the operating frequency (assuming that there are no static bias currents in the microprocessor core) [6]. Let us assume that a subroutine takes Δt time to execute. This implies that the energy consumed by a single execution of the subroutine is

$$E_{tot} = P_{tot}\Delta t = C_{tot}V_{dd}^2 + V_{dd}I_{leak}\Delta t \quad (7)$$

where C_{tot} is the total capacitance switched by executing subroutine. Clearly, if the execution time of the subroutine is changed (by changing the clock frequency), the total switched capacitance, C_{tot}, remains the same. Essentially, the integrated circuit goes through the same set of transitions except that they occur at a slower rate. Therefore, if we execute the same subroutine at different frequencies, but at the same voltage, and measure the energy consumption we should observe a linear increase with the execution time with the slope being proportional to the amount of leakage.

4.2 Observations

The subroutine chosen for execution was the decimation-in-time Fast Fourier Transform (FFT) algorithm because it is a very standard, computationally intensive, Digital Signal Processing (DSP) operation. The microprocessor, therefore, runs at maximum 'horsepower'. The execution time for an $N = 1024$ point FFT on the StrongARM is a few tenths of a second and scales as $O(N\log N)$. To obtain accurate execution time and stable current readings, the FFT routine was run a few hundred times for each observation. A total of eighty different data points corresponding to different supply voltages between 0.8 V and 1.5 V and operating frequencies between 59 MHz and 206 MHz were compiled.

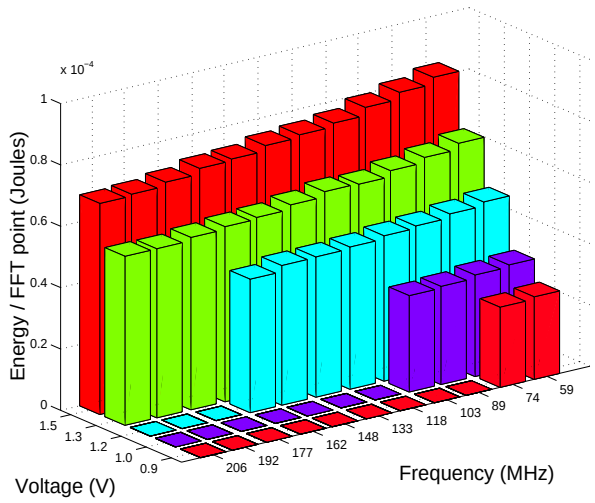

Figure 7. FFT energy consumption

Figure 7 illustrates the implications of Equation 7. When the operating frequency is fixed and the supply voltage is scaled, the energy scales almost quadratically. On the other hand, when the supply voltage is fixed and the frequency is varied the energy consumption decreases linearly with frequency (i.e. increases linearly with the execution time) as predicted by Equation 7. Not all frequency, voltage combinations are possible. For example the maximum frequency of the StrongARM is 206 MHz and it requires a minimum operating voltage of 1.4 V.

We can measure the leakage current from the slope of the energy characteristics, for constant voltage operation. One way to look at the energy consumption is to measure the amount of charge that flows across a given potential. The charge attributed to the switched capacitance should be independent of the execution time, for a given operating voltage, while the leakage charge should increase linearly with the execution time. Figure 8 shows the measured charge flow as a function of the operating frequency for a 1024 point FFT. The amount of charge flow is simply the product of the execution time and current drawn. As expected, the total charge consumption decreases almost linearly with operating frequency (i.e. increases with execution time) and the slope of the curve, at a given voltage, directly gives the leakage current at that voltage.

The dotted lines are the linear fits to the experimental data in the minimum mean-square error sense. At this point it is worthwhile to mention that the term "leakage current" has been used in an approximate sense. Truly speaking, what we are measuring is the total static current in the processor, which is the sum of leakage and bias currents. However, in the SA-1100 core, the bias currents are small and most of the static currents can be attributed to leakage. This assertion is further supported by the fact that the static current we measure has an exponential behavior as shown in the next section.

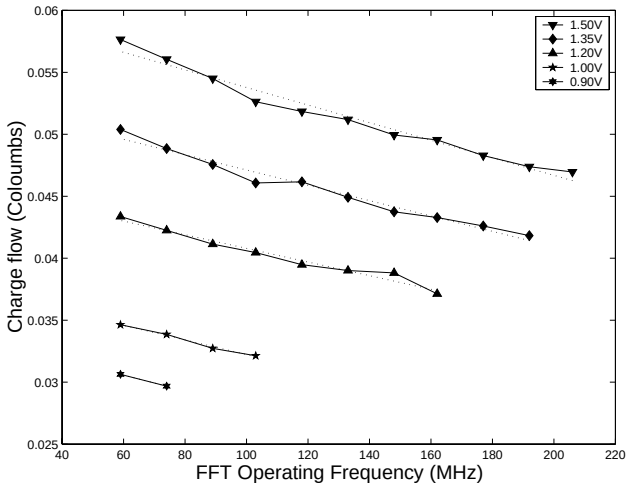

Figure 8. FFT charge consumption

From the BSIM2 MOS transistor model [7], the sub-threshold current in a MOSFET is given by

$$I_{sub} = Ae^{\frac{(V_G - V_S - V_{TH0} - \gamma' V_S + \eta V_{DS})}{n' V_T}}\left(1 - e^{-\frac{V_{DS}}{V_T}}\right) \quad (8)$$

where

$$A = \mu_0 C_{ox}\frac{W_{eff}}{L_{eff}}V_T^2 e^{1.8} \quad (9)$$

and V_T is the thermal voltage, V_{TH0} is the zero bias threshold voltage, γ' is the linearized body effect coefficient, η is the Drain Induced Barrier Lowering (DIBL) coefficient and V_G, V_S and V_{DS} are the usual gate, source and drain-source voltages respectively. The important point to observe is that the subthreshold leakage current scales exponentially with the drain-source voltage.

The leakage current at different operating voltages was measured as described earlier, and is plotted in Figure 9. The overall microprocessor leakage current scales exponentially with the supply voltage. Based on these measurements the following model for the overall leakage current is proposed for the microprocessor core,

$$I_{leak} = I_0 e^{\frac{V_{dd}}{nV_T}} \quad (10)$$

where I_0 = 1.196 mA and n = 21.26 for the StrongARM SA-1100.

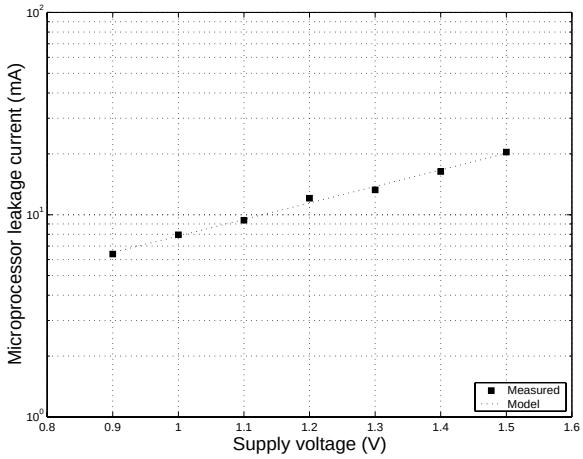

Figure 9. Leakage current variation

4.3 Explanation of exponential behavior

The exponential dependence of the leakage current on the supply voltage can be attributed to the DIBL effect. Consider the stack of NMOS devices shown in Figure 10. Equation 8 suggests that for a single transistor, the leakage current should scale exponentially with V_{DS}=V_{DD} because of the DIBL effect. However since the ratio V_{DS} / V_T is larger than 2, the term inside the brackets of Equation 8 is almost 1. It has been shown in [8] that this approximation is also true for a stack of two transistors. With three or more transistors, the ratio V_{DS} / V_T for at least the lowest transistor becomes comparable to or even less than 1. Therefore, the term inside the bracket of Equation 8 cannot be neglected for such cases. The leakage current progressively decreases as the number of transistors in the stack increases and for a stack of more than three transistors the leakage current is small and can be neglected. It has further been shown in [8] that the ratio of the leakage currents for the three cases shown in Figure 10 can be written as

$$I_{l1} : I_{l2} : I_{l3} = 1.8e^{\frac{\eta V_{DD}}{nV_T}} : 1.8 : 1 \quad (11)$$

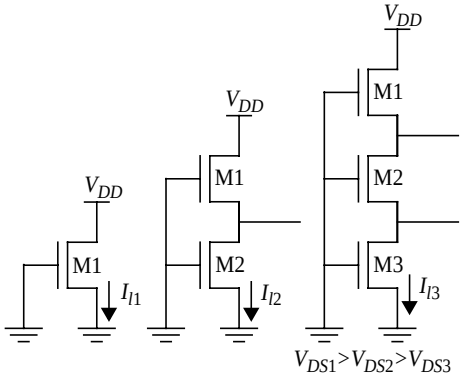

Figure 10. Effect of transistor stacking

Therefore, the leakage current of a MOS network can be expressed as a function of a single MOS transistor (by accounting for the signal probabilities at various nodes and using the result of Equation 11). If the number of stacked devices is more than three, the leakage current contribution from that portion of the circuit is negligible. If there are three transistors stacked such that two of them are 'OFF' and one is 'ON' then the leakage analysis is the same as the stack of two 'OFF' transistors. For parallel transistors, the leakage current is simply the sum of individual transistor leakages. A similar argument holds for PMOS devices. Since, the leakage current of a single MOS transistor scales exponentially with V_{DD}, using the above arguments, we can conclude that the total microprocessor leakage current also scales exponentially with the supply voltage.

4.4 Separation of current components

Table 2 compares the measured leakage current with the values predicted by Equation 5. The maximum percentage error measured was less than 6% over the entire operating voltage range of the StrongARM which suggests a fairly robust model.

TABLE 2 : LEAKAGE CURRENT MEASUREMENTS

V_{DD} (V)	I_{leak} (mA)		Error (%)
	Measured	Model	
1.50	20.41	20.10	1.50
1.40	16.35	16.65	-1.84
1.30	13.26	13.80	-4.04
1.20	12.07	11.43	5.27
1.10	9.39	9.47	-0.87
1.00	7.96	7.85	1.40
0.90	6.39	6.53	-1.70

Based on the leakage model described by Equation 10, the static and dynamic components of the microprocessor current consumption can be separated. The standby current of the StrongARM in the "idle" mode at 1.5 V is about 40 mA. This is not just the leakage current but also has the switching current due to the circuits that are still being clocked. On the other hand, this technique neatly separates the pure leakage component (assuming negligible static currents) from all other switching currents.

5. RESULTS

The estimation techniques described in the previous sections were implemented in a web-based tool called JouleTrack. The tool is available at http://dry-martini.mit.edu/JouleTrack. The broad approach in the tool is summarized in Figure 11. The user uploads his C source code. The webserver uses Common Gateway Interface (CGI) scripts to create a temporary workarea for the user. His programs are compiled and linked with any standard C libraries. The user also specifies any command line arguments that the program might need along with a target operating frequency. Compiler optimization options are also available. The user can choose the current prediction model. Compile/link time errors are reported back by the CGI to the user. If no errors exist the program is executed on an ARM simulator which produces the program outputs (which the user can view), assembly listing

(which can also be viewed) as well as run-time statistics like execution time, cycle counts etc. These statistics are fed into an estimation engine which computes the energy profile and charts the various energy components using the methodology described in the previous sections.

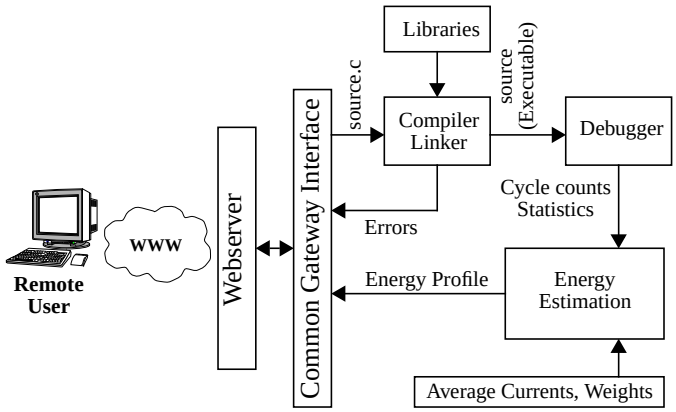

Figure 11. JouleTrack block diagram

6. CONCLUSIONS

Based on experiments done on the StrongARM and Hitachi SH-4 processors we conclude that the common overheads present across instructions result in the variation in current consumption of different instructions being small. The variation in current consumption of programs is even smaller (less than 8% for the benchmark programs that we tried). Therefore, to a first order, we can assume that current consumption depends only on operating frequency and supply voltage and is independent of the executing program. A second order model that uses energy differentiated instruction classes/cycles is also proposed and it was shown that the resulting current prediction error was reduced to less than 2%. A methodology for separating the leakage and switching energy components is also discussed. The proposed leakage current model has less than 6% error. The techniques have been implemented in an web-based software energy estimation tool called JouleTrack.

ACKNOWLEDGMENTS

This research is sponsored by the Defense Advanced Research Project Agency (DARPA) Power Aware Computing/Communication Program and the Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0551.

REFERENCES

[1] V. Tiwari, S. Malik and A. Wolfe, “Power Analysis of Embedded Software: A First Step Towards Software Power Minimization”, IEEE Transactions on VLSI Systems, vol. 2, no. 4, Dec. 1994, pp. 437-445

[2] J. T. Russell, M. F. Jacome, “Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors”, Proc. of the International Conference on Computer Design, 1998, pp. 328-333

[3] H. Mehta, R. M. Owens, M. J. Irwin, “Instruction Level Power Profiling”, ICASSP’96, pp. 3326-3329

[4] Intel StrongARM Processors, http://developer.intel.com/design/strong/sa1100.htm

[5] Hitachi SuperH Processors, http://www.superh.com/

[6] J. M. Rabaey, *Digital Integrated Circuits : A Design Perspective*, Prentice Hall, New Jersey, 1996

[7] J. Sheu, et. al., “BSIM: Berkeley short-channel IGFET model for MOS transistors”, IEEE Journal of Solid-State Circuits, SC-22, 1987

[8] R. X. Gu and M. I. Elmasry, “Power dissipation analysis and optimization of deep submicron CMOS digital circuits”, IEEE Journal of Solid State Circuits, vol. 31, no. 5, May 1996, 707-713