Energy Aware Software

Amit Sinha
Department of EECS
Massachusetts Institute of Technology
Cambridge, MA 02139
Tel : 1-617-253-0164
e-mail : sinha@mit.edu

Anantha P. Chandrakasan
Department of EECS
Massachusetts Institute of Technology
Cambridge, MA 02139
Tel : 1-617-228-7619
e-mail : anantha@mtl.mit.edu

Abstract - The concept of energy aware software is introduced. A simple energy model for software is presented that separates the switching and leakage components and predicts its total energy consumption with less than 5% error for a set of benchmark programs. The experiments have been performed on the StrongARM SA-1100 microprocessor. A mathematical model for the total leakage current has also been proposed and it has been shown that they can account for about 10% of the energy dissipation for low threshold voltage microprocessors and, assuming continuous operation, the leakage energy fraction gets significantly higher for lower duty cycle

I. Introduction

As wireless communications goes into the realm of network-based services such as wireless internet access and video phones, to name a few, the constraints on energy efficiency of portable, battery-operated, hand-held systems would significantly increase. Personal communication products based on Java-enabled digital signal processing have already been proposed. Such systems would not have all applications resident in them but would derive functionality from the wireless network or the internet. Applications would have to be downloaded from the network upon request.

The energy constrained hand-held devices should be able to estimate the energy requirement of an application that has to be executed and make subsequent decisions about its processing ability based on user-input and sustainable battery life. Therefore, the concept of 'energy aware' software is integral to such systems. For example, based on the energy model for the application, the system should be able to decide whether the particular application can run at the desired throughput. If not, it might be able to reduce its voltage using an embedded DC/DC converter and run the application at reduced throughput [1]. On the other hand, if the user desires he could have the application running at the same throughput but reduced accuracy [2]. These energy-accuracy-throughput tradeoffs necessitate robust energy models for software based on parameters such as operating frequency, voltage and target processor. This idea is summarized in Fig. 1. The 'function' or application that is downloaded has an associated energy model which the portable system can use to configure itself based on user specification and energy availability.

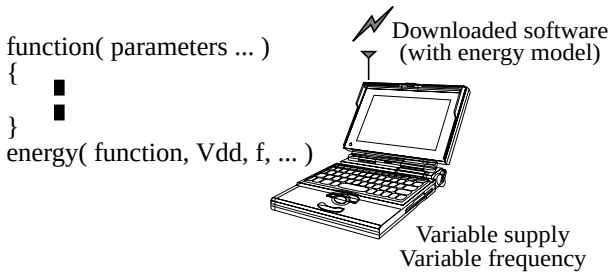


Fig. 1. Energy aware software

Another domain of interest would be distributed sensor applications [3] where, once again, the energy constrained sensors should be able to configure themselves based on their current energy status and energy models for the task resident or transmitted to them. Clustering schemes and transmission mechanisms would rely on such predictive models.

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. Multiple thresholds have been used to deal with the leakage problem [4][5].

In this paper we propose a simple energy model for software based on frequency and supply voltage as parameters. The model incorporates explicit characterization of leakage energy as opposed to switching energy. A technique to isolate the two components has been demonstrated. The relative significance of these energy components under different duty cycle tasks has also been explored.

II. Experimental setup

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. The SA-1100 consists of a 32-bit RISC processor

core, with a 16 KB instruction cache and an 8 KB write-back data cache, a minicache, a write buffer, and a Memory Management Unit (MMU) combined in a single chip. 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. The I/O pads run 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. Software power estimation techniques, have been explored in literature [8]. As will be shown subsequently, the leakage measurement technique relies on measuring the charge consumption by a subroutine. To measure the charge we need to know the current drawn while the subroutine is executing and the exact execution time. The execution time for multiple iterations was obtained accurately using the time utility in C and the execution time per iteration and charge consumption were subsequently computed.

The core supply voltage was altered directly from the external supply while the internal clock frequency of the processor was changed via software control. Details of the StrongARM setup can be found in [11][12].

III. Energy modelling

A. Principle

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

$$P_{tot} = P_{dyn} + P_{stat} = C_L V_{dd}^2 f + V_{dd} I_{leak} \qquad (1)$$

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) [7]. 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

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

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.

B. Observations

The subroutine chosen for execution was the decimation-in-time Fast Fourier Transform (FFT) algorithm [9] 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.

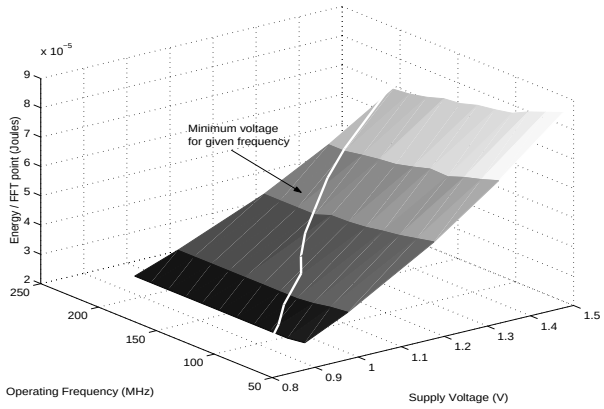


Fig. 2. FFT energy consumption

Fig. 2 illustrates the implications of Equation 2. 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 2. 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. The line across the surface plot demarcates the possible operating regions from the extrapolated ones (i.e. the minimum operating voltage for a given frequency).

IV. Leakage energy model

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. Fig. 3 shows the measured charge flow as a function of the execution time 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 increases almost linearly with execution time and the slope of the curve, at a given voltage, directly gives the leakage current at that voltage.

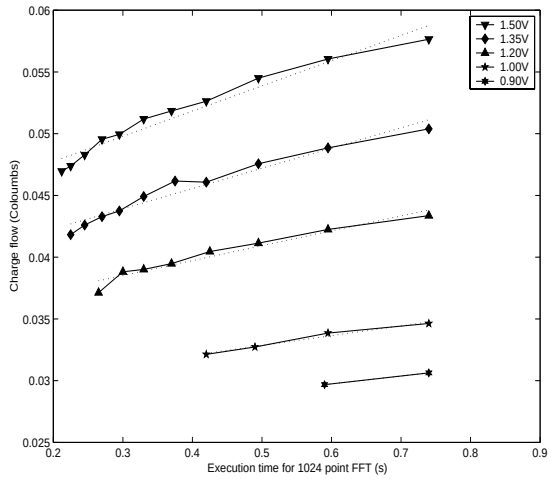


Fig. 3. FFT charge consumption

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.

From the BSIM2 MOS transistor model [6], the subthreshold current in a MOSFET is given by

$$I_{sub} = Ae^{\frac{(V_G - V_S - V_{TH0} - \gamma' V_S + \eta V_{DS})}{n' V_T}} \left(1 - e^{-\frac{V_{DS}}{V_T}}\right) \quad (3)$$

where

$$A = \mu_0 C_{ox} \frac{W_{eff}}{L_{eff}} V_T^2 e^{1.8} \quad (4)$$

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 Fig. 4. 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,

$$I_{leak} = I_0 e^{\frac{V_{dd}}{n V_T}} \quad (5)$$

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

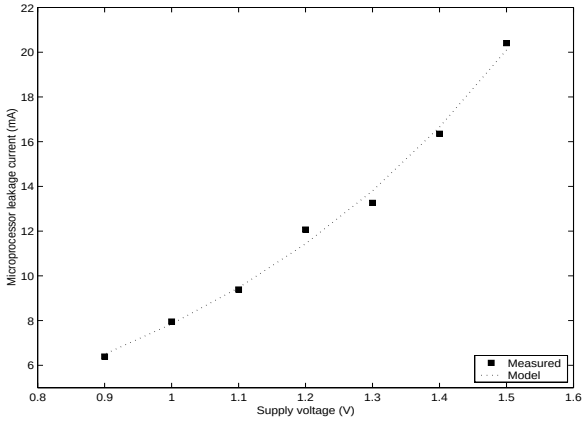


Fig. 4. Leakage current variation

A. 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 Fig. 5. Equation 3 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 3 is almost 1. It has been shown in [10] 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 3 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 [10] that the ratio of the leakage currents for the three cases shown in Fig. 5 can be written as

$$I_{l1} : I_{l2} : I_{l3} = 1.8e^{\frac{\eta V_{DD}}{n V_T}} : 1.8 : 1 \quad (6)$$

Therefore, the leakage current of a MOS network can be expressed as a function a single MOS transistor (by accounting for the signal probabilities at various nodes and using the result of Equation 6). 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.

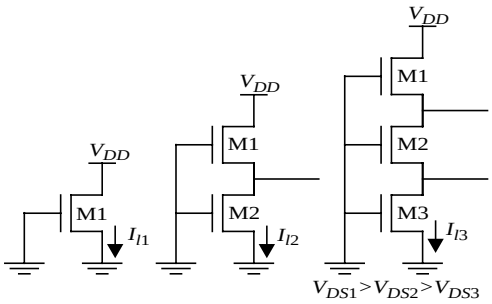


Fig. 5. Effect of transistor stacking

B. Separation of current components

Table I 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 I : 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 5, the static and dynamic components of the microprocessor current consumption were separated. These currents have been plotted in Fig. 6. The operating frequency at each voltage was chosen to be the maximum possible for that voltage. 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.

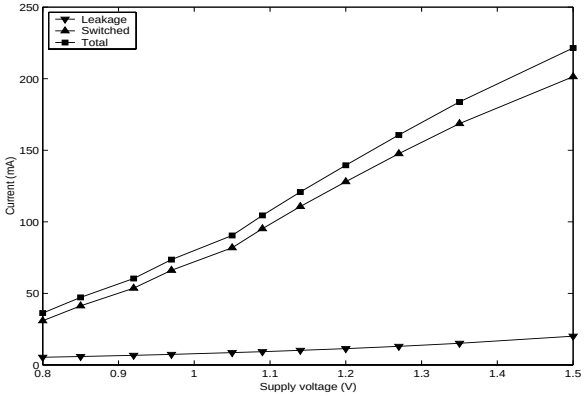


Fig. 6. Static, dynamic and total current

For low threshold microprocessors like the StrongARM, it can seen that the leakage current is quite substantial (about 10% in this case). The leakage current, as a fraction of the total current can be expressed as

$$\beta_{leak} = \frac{I_0 e^{\frac{V_{dd}}{nV_T}}}{I_0 e^{\frac{V_{dd}}{nV_T}} + kV_{dd}^{\alpha}} \quad (7)$$

where kV_{dd}^{α} models the switching current, has a very interesting profile. The leakage component, as a fraction of the total current, can be shown to have a minima at $V_{dd} = n\alpha V_T$, if we use Equation 7. For the StrongARM, this is about 1.2 V as shown in Fig. 7.

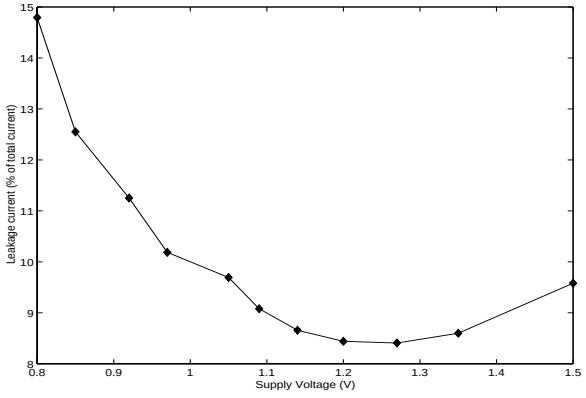


Fig. 7. Leakage current fraction

V. ESTIMATION OF SWITCHING ENERGY

Once the leakage current is known, we can estimate the leakage energy from the knowledge of the supply voltage and execution time. The switching component can subsequently be obtained simply by subtracting the leakage energy from the total energy as suggested by Equation 2. The total switched capacitance and be obtained by dividing out the supply volt-

age from the switching energy. Fig. 8 plots the total switched capacitance and the total number of execution cycles (for 80 FFT executions at different voltage frequency combinations) as a function of supply voltage.

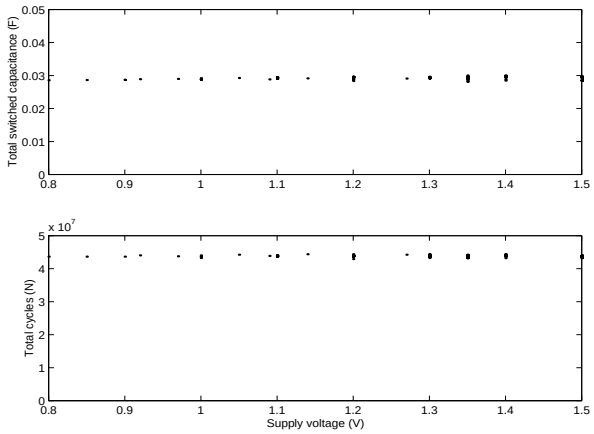


Fig. 8. Switched capacitance and total cycles

The total number of cycles, as measured from the execution time and operating frequency, is fixed (since we are executing the same program). However, the total switched capacitance also turns out to be independent of the supply voltage within the limits of experimental accuracy. The total average switched capacitance for the 1024 point FFT was 0.0292 F. The total switched capacitance is obviously program dependent. However, the total switched capacitance per cycle, on the StrongARM has variation of about 20% depending on the type of instruction, cache misses etc.

VI. Energy Tradeoffs

As the supply voltages and thresholds are reduced, system designers have to pay increasing attention to leakage currents. For the StrongARM, at maximum duty cycle and minimum voltage (for a given frequency), the leakage energy is about 10%. However, the leakage energy rises exponentially with supply voltage and decreases linearly with frequency as shown in Fig. 9. Therefore, operating at a voltage, above the minimum possible, for a given frequency, is not advisable. This might be an issue in fixed supply, variable frequency systems. For low duty-cycle systems, the overall energy consumption becomes increasingly dominated by leakage effects. The fixed task consumes a certain amount of switching energy per execution while the system leaks during the idle mode between tasks. Extensive clock gating techniques, such as those present in the StrongARM, reduce the unnecessary switching energy in the "idle" mode. The StrongARM does also have a "sleep" mode where the supply voltage is reduced to zero for most circuits, and the processor state is stored. This significantly reduces the leakage problem. However, reverting to sleep mode between duty cycles may incur a lot of overhead (in terms of cycles and energy) or may not be supported by the target processor.

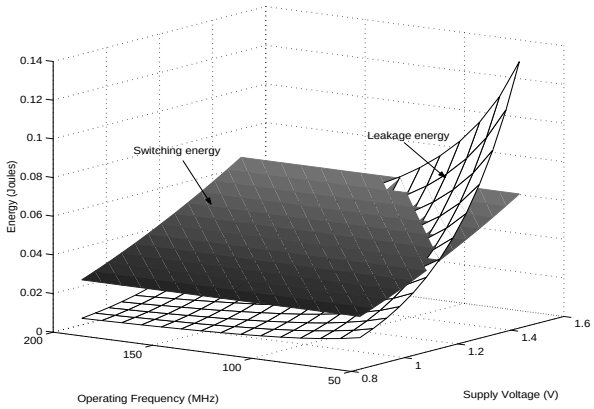
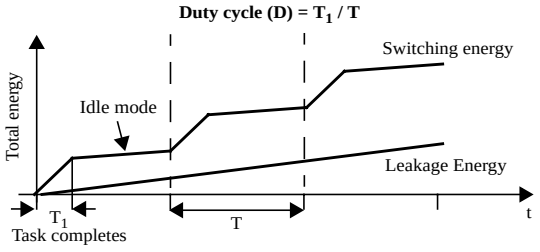
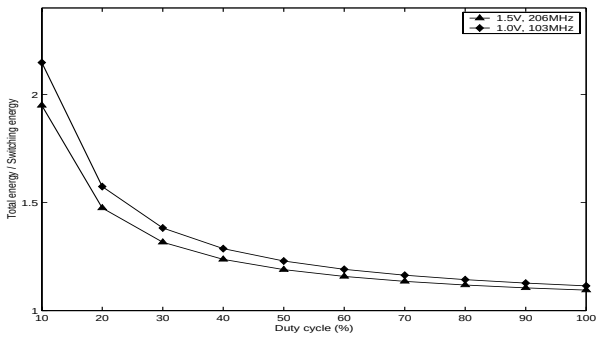


Fig. 9. FFT energy components

Fig. 10 illustrates the effect of duty cycle on the energy consumption of a system. Suppose the system has to do an FFT every T seconds such that the execution time for the FFT is $T_1 \leq T$. After computing the FFT, the processor enters "idle" mode and switching activity is reduced by clock gating techniques. Leakage on the other hand is unaffected. Fig. 10 (b) plots the ratio of total energy consumption to the switching energy, as a function of duty cycle. For a low duty cycle of 10% the ratio is about 2 for our FFT data i.e. almost twice the amount of energy is used up for the same task compared to the 100% duty cycle case.



(a)



(b)

Fig. 10. Low duty cycle effects

VII. Results

The following parameterized model for software has been proposed and verified on the StrongARM SA-1100 microprocessor,

$$E_{tot}(V_{dd}, f) = C_{tot}V_{dd}^2 + V_{dd}\left(I_0 e^{\frac{V_{dd}}{nV_T}}\right)\left(\frac{N}{f}\right) \qquad (8)$$

where C_{tot} is the total capacitance switched by the program, and N is the number of cycles the program takes to execute. Both these parameters can be obtained from the energy consumption data for a particular supply voltage, V_{dd}, and frequency, f, combination. The model can then be used to predict energy consumption for different supply-throughput configurations in energy constrained environments. The leakage current model is processor dependent. Using the technique described in Section IV, one can determine the parameters I_0 and n. For the StrongARM SA-1100 microprocessor these parameters were 1.196 mA and 21.26 respectively.

Table II shows the performance of our model compared to actual energy data for six standard programs. The maximum error for the programs we tested was less than 5%. For the StrongARM the current consumption of different instructions ranges from 170 mA to 230 mA. This implies that the switched capacitance per cycle can vary as much as 25%. However, for sufficiently long programs (with $N > 10^6$ cycles say), which require both integer and floating point operations, typically these instruction level variations tend to get averaged out. For instance, the switched capacitance per cycle in Table II is close to 0.67 nF for all the programs.

Table II : Model performance

Program	Energy (mJ)	C_{tot} (mF)	N (x10^6)	$\frac{C_{tot}}{N}$	Error (%)
fft	53.89	28.46	43.67	0.65	1.24
dct	0.10	0.05	0.08	0.66	4.22
idct	0.13	0.06	0.10	0.66	2.59
fir	1.23	0.67	0.97	0.70	3.28
log	4.54	2.46	3.71	0.67	3.94
tdlms	21.29	12.13	17.10	0.71	1.91

VIII. Conclusions

The concept of energy aware software has been introduced which incorporates an energy model with the corresponding application. Based on experiments conducted on the StrongARM SA-1100 microprocessor, a software energy model has been proposed which separates the leakage and switching energy components. The model showed less than 5% prediction error for the programs that were tested. A macro leakage current model, to estimate the overall microprocessor leakage current has also been proposed. This model has less than 6% error from experimental data. Based on our experiments we conclude that as supply voltages and thresholds scale, leakage (static) components will become increasingly dominant. For the StrongARM, at 100% duty cycle, the leakage energy is about 10% and increases exponentially with supply voltage and decreases linearly with operating frequency.

Acknowledgements

This work was supported by an NSF Career Development Award MIP-9501995. The authors would like to acknowledge valuable help from Gangadhar Konduri and Alice Wang with the StrongARM setup.

References

[1] V. Gutnik and A. Chandrakasan, "An embedded power supply for low-power DSP", IEEE Trans. on VLSI Systems, vol. 5, no. 4, Dec. 1997, pp. 425-435

[2] S. H. Nawab, et. al., "Approximate signal processing", Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, vol. 15, no. 1/2, Jan. 1997, pp. 177-200

[3] K. Bult, et. al., "Low power systems for wireless microsensors", IEEE/ACM International Symposium on Low Power Electronics and Design, Aug. 1996, pp. 17-21

[4] J. Kao, S. Narendra and A. P. Chandrakasan, "MTCMOS hierarchical sizing based on mutual exclusive discharge patterns", Proceedings of the 35th Design Automation Conference, San Francisco, June 1998, pp. 495-500

[5] L. Wei, et. al., "Design and optimization of low voltage high performance dual threshold CMOS circuits", Proceedings of the 35th Design Automation Conference, San Francisco, June 1998, pp. 489-494

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

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

[8] V. Tiwari and S. Malik, "Power analysis of embedded software: A first approach to software power minimization", IEEE Trans. on VLSI Systems, vol. 2, Dec. 1994

[9] A. V. Oppenheim and R. W. Schafer, *Discrete Time Signal Processing*, Prentice Hall, New Jersey, 1989

[10] 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

[11] Advanced RISC Machines Ltd., *Advance RISC Machines Architectural Reference Manual*, Prentice Hall, New York, 1996

[12] Advanced RISC Machines Ltd., *ARM Software Development Toolkit Version 2.11 : User Guide*, May 1997