Energy Aware Software

by

Amit Sinha

Bachelor of Technology in Electrical Engineering
Indian Institute of Technology, Delhi, 1998

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

Master of Science
in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

December 1999

Author .....
Department of Electrical Engineering and Computer Science
December, 1999

Certified by .....
Anantha Chandrakasan
Associate Professor
Thesis Supervisor

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

Energy Aware Software

by

Amit Sinha

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

Abstract

As wireless communications goes into the realm of network-based services such as wireless internet access and video phones, that may derive their functionality from the wireless network or the internet, the energy constraints on the portable devices would significantly increase. The systems running these applications should be aware of the energy cost of these applications and make subsequent decisions about their processing ability based on user-input and sustainable battery life. Scalable algorithms that provide efficient tradeoffs between energy, accuracy and throughput will have to be developed. As threshold voltage on processors reduce, leakage energy is also becoming a significant problem. This work presents an energy model for software based on experiments performed on the StrongARM microprocessor. The proposed energy model has less than 5% error from experimental values. Energy scalability of algorithms, in both software and hardware, has also been investigated. It has been demonstrated that restructuring computation (both at the software and hardware abstractions) can result in efficient energy-quality scalability of the algorithm which can be very desirable in energy constrained systems.

Thesis Supervisor: Anantha Chandrakasan
Title: Associate Professor

Acknowledgements

I would like to express my sincere gratitude to Prof. Anantha P. Chandrakasan. I met him twice before I came to MIT and I was convinced that working with him would be a most stimulating and rewarding experience, which now in retrospect, seems to be one of the best decisions I ever made. His ideas and suggestions, keen insight, sense of humor, never-say-die enthusiasm and perennial accessibility have been instrumental in getting this work done. Apart from academic counsel, I consider his professional accomplishments and career a model to emulate.

I would also like to thank all my colleagues in 38-107. Jim Goodman for his unbelievable patience in answering all my questions about the StrongARM and Framemaker. Alice Wang for being the perfect effervescent colleague, sharing her MIT expertise, the foodtruck lunches, and always being there to help. Thanks to her I managed to sail across the rough waters of 6.374 as a TA. Manish Bhardwaj and Rex Min for burning midnight oil with me, not to forget the numerous wagers that were thrown in the wee hours of the morning! During these nocturnal discussions the meaning of life and the cosmic truths were discovered many times over! Special thanks to Gangadhar Konduri for helping me get settled when I first came to MIT. His widespread *desi* network got my social life going. Wendi, who's spotless desk and meticulous organization always made me sick, for helping me with her video expertise. SeongHwan, for his reassuring presence and for making TAing 6.374 so much less grungy. Travis Furrer for his perpetual keenness to help whether it was the idiosyncrasies of Java or the installation of a painful Solaris patch. Eugene Shih, Paul Peter, Vadim Gutnik, Scott Meninger, Joshua Bretz and Oscar Mur-Miranda for being such great and helpful officemates. I would also like to take this opportunity to thank the veterans - Raj, Duke and Tom Simon for helping me despite their tapeout deadlines and teaching me how to take a more critical and less gullible view of things.

I wish to thank all my friends at MIT and especially my apartment mates Ashish Kelkar and Atul Dalmia for making the home-on-the-dome work out for me. Thanks to Ashish my palate was

not totally deprived of fine Indian cuisine! Together we roller coasted the ups and downs of graduate life and refused to drink water from the fire hose!

Finally, I would like to thank the people who are dearest to me and without who's love and support none of this would have been possible. Deepali, for her patience, love and encouragement. My mother, Mrs. Anjali Sinha, my father Mr. R. N. Sinha, and my sister, Neha, to whom I owe everything. Their blessings, love and sacrifices has made me what I am today.

Amit Sinha

MIT, December 1999

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Energy efficient system design is becoming increasingly important with the proliferation of portable, battery-operated devices such as laptops, Personal Digital Assistants (PDAs) and cellular phones. Energy constraints on these devices are becoming increasingly tight as complexity and performance requirements continue to be pushed by user demand. While complexity and integration density of digital systems has been growing exponentially the parallel growth in battery technology has been very incremental [1]. Therefore while the microprocessor power consumption has gone up from under 1 Watt to over 100 Watts over the last 20 years, the corresponding improvement in battery technology has been less than a factor of 4 over the last three decades. Recent advances in processor technology for portable applications have however reduced power consumption to a few watts.

Reducing power consumption is also an important design objective in stationary desktop equipment because of the increasing cost associated with complex cooling systems and packaging [2]. Concerns about the environmental cost of electronic systems also drive the trend towards energy efficient design. Certain reliability issues also directly relate to power consumption in on-chip circuits. Electromigration is one such issue which results in short-circuits and breaks when atoms in the metal lines migrate under high electric fields. Large on-chip currents exacerbate power distribution problems due to increased resistive drops, ground bounce and inductive effects. There is a difference between power and energy consumption. While the primary concern in desktop equipment is peak power consumption (from a packaging, cooling and reliability perspective), average power consumption, i.e. energy, is of primary concern in portable equipment because it directly translates to battery life.

Numerous circuit and computer-aided design techniques have been proposed for low power [3]. The power consumption of a CMOS circuit can be represented as

$$P = \alpha C_L V_{dd}^2 f \quad (1\text{-}1)$$

where C_L is the total physical capacitance, V_{dd} is the power supply voltage, f is the frequency and α is the activity factor of the circuit. Almost all low power circuit innovations can be categorized into techniques that reduce one of the above parameters. However, the ever increasing integration densities (more total load capacitance) and performance requirements (higher frequency) have resulted in the overall power consumption increasing to a point where thermal packaging limits are being reached. Increasing performance requirements dictate that the supply voltage be high (since the speed of the circuit approximately varies directly with V_{dd}) while the power consumption increases quadratically with V_{dd}. Thus a trade-off is involved in choosing a very low voltage supply. Clock gating, where idle portions of the circuit are shut off, has been used effectively for low power.

It has been shown in separate applications that dedicated hardware implementations can out perform general purpose microprocessors/DSPs by several orders of magnitude in terms of energy consumption [4][5][6]. However, dedicated implementations are not always feasible. The prohibitive cost of fabrication lines make only high volume parts such as processors, memories and FPGA's economically feasible. Therefore, Application Specific Integrated Circuits (ASICs) are getting increasingly expensive to manufacture and are a solution only when speed or power constraints dictate otherwise. Furthermore, introducing revisions and changes into hardwired solutions is expensive and time-consuming. The breaking of the $5 threshold for 32-bit processors has resulted in an explosion in the use of general purpose microprocessors and DSPs in high-volume embedded applications [7].

Another factor that has influenced the shift towards programmable software solutions is the ever shrinking time-to-market requirement. The presence of powerful and mature software development environments as well as the economics involved in having programmable solutions on general purpose processors rather than hardwired ones, have further contributed to this trend.

1.1 Energy Aware Software

Pure hardware optimizations can no doubt decrease the power consumption significantly. However, it the software that finally runs on the hardware platform and therefore the overall system power consumption significantly depends on software. Further, the choice of the algorithm can have an enormous impact on the energy consumption. An inefficient algorithm, can result in significantly higher switched capacitance and execution time and therefore more energy per useful computation. Energy aware software is defined as algorithms and the corresponding code that can account for its energy consumption and can be used to improve the overall energy efficiency of the system. The idea being to test and model software energy consumption and write efficient code than minimizes system energy. Software energy awareness could then be used to come up with compilers and CAD tools that map an algorithm into an energy efficient code for a specific target system. Energy aware software can be used to predict battery life based on the energy consumption model and current battery state e.g. a user on a long journey might choose to trade-off some quality/performance and extend the battery life of his laptop.

1.2 Algorithmic Transforms for Energy Scalable Computation

In embedded systems energy is a precious resource and must be used efficiently. Therefore, it is highly desirable that we structure our algorithms and systems in such a fashion that computational accuracy can be traded off with energy requirement. At the heart of such transformations lies the concept of *incremental refinement* [8]. Consider a full-motion video telephone. It would be highly desirable to restructure the image decoding steps in an incremental and energy scalable fashion, e.g., if the user feels that the conversation is going to take place for a longer duration than his battery can sustain, he should be able to double his battery life by reducing the energy consumption per frame to half while still having acceptable picture quality. Investigation of such energy scalable algorithms for various signal processing and general purpose computing applications would be very helpful for maximum energy utility in portable systems.

1.3 Contributions of the Thesis

The major contribution of this thesis has been investigating the energy consumption of programmable/embedded software solutions where energy is a precious resource. The first part of this thesis (Chapter 2) deals with modeling software energy consumption at a macro level. 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 second part of this thesis (Chapter 3) deals with algorithmic transforms that aid in extracting the maximum output quality from an algorithm for a given energy availability. We introduce the notion of energy scalable computation on general purpose processors. The desirable energy-quality behavior of algorithms is discussed. Subsequently the energy-quality scalability of two distinct categories of commonly used signal processing algorithms (viz. filtering and frequency domain transforms) are analyzed on the StrongARM SA-1100 processor and transformations are described which obtain significant improvements in the energy-quality scalability of the algorithm.

However, to obtain overall system energy scalability, all stages need to be scalable. Algorithmic transforms can ensure a certain energy-quality performance but if the datapath of the processor on which the algorithm runs can be scaled based on a *precision-on-demand* type approach, the system energy scalability can be dramatically improved. The third part of this thesis (Chapter 4) deals with such energy scalable hardware architectures. We present a novel Finite Impulse Response (FIR) filter architecture based on a Distributed Arithmetic (DA) approach with two supply voltages and variable bit precision operation. The filter is able to adapt itself to the minimum bit precision required by the incoming data and also operate at a lower voltage so that it still meets a fixed throughput constraint. As opposed to the worst case fixed precision design, our precision-on-demand implementation has an energy requirement that varies linearly with the average bit precision required by the input signal. We also demonstrate that 50% to 60% energy savings can easily be obtained in the case of speech data.

Chapter 2
Software Energy Modeling

2.1 Motivation

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 with increased latency [10]. On the other hand, if the user desires he could have the application running at the same throughput but reduced accuracy. These energy-accuracy-throughput trade-offs necessitate robust energy models for software based on parameters such as operating frequency, voltage, accuracy required, target processor etc. This idea is summarized in Figure 2-1. 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, e.g., the operating system running on a laptop can compute the expected battery lifetime based on the energy model for the given application and prompt the user accordingly.

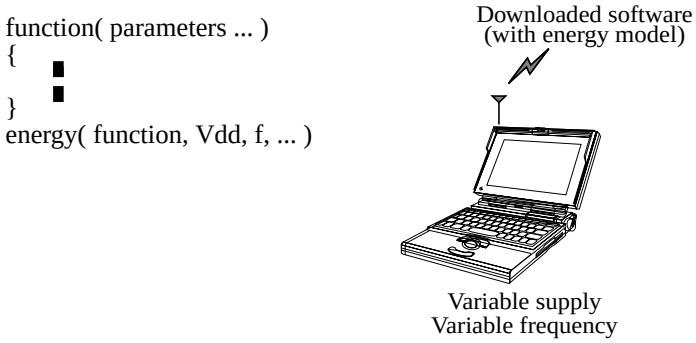


Figure 2-1: Energy aware software

Another domain of interest would be distributed sensor applications [11][12] 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 [13][14].

In this chapter, 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.

2.2 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.

Figure 2-2: StrongARM SA-1100 experimental setup

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. This does not account for the power consumption in the external memory which runs from an embedded fixed power supply. Software power estimation techniques, have been explored in literature [16]. 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 [17][18].

2.3 Energy Modeling

2.3.1 Principle

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

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

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)

. 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

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

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.

2.3.2 Observations

The subroutine chosen for execution was the decimation-in-time Fast Fourier Transform (FFT) algorithm [20] because it is a very standard, computationally intensive, Digital Signal Processing (DSP) operation. 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. (See Appendix A)

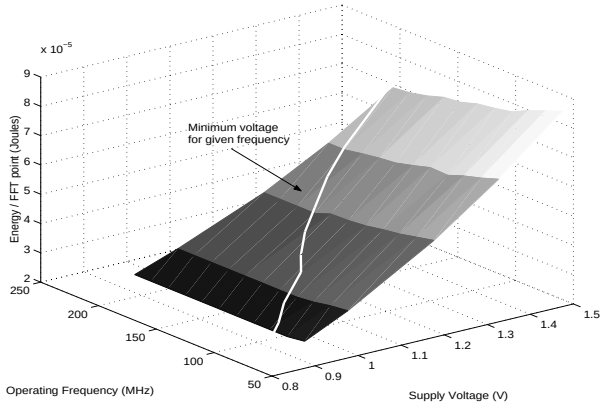


Figure 2-3: FFT energy consumption

Figure 2-3 illustrates the implications of Equation 2-2. 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 2-2. 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).

2.4 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. Figure 2-4 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.

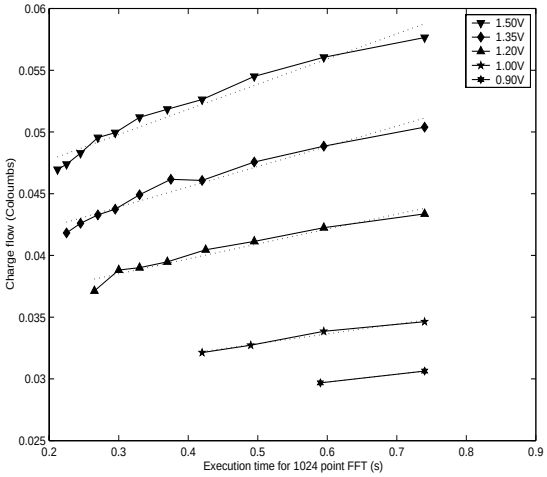


Figure 2-4: FFT charge consumption

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 [21][22]. 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 [15], the sub-threshold current in a MOSFET is given by

$$I_{sub} = Ae^{\frac{(V_G - V_S - V_{TH0} - \gamma' V_S + \eta V_{DS})}{n' V_T}} \left(1 - e^{-\frac{V_{DS}}{V_T}}\right) \tag{2-3}$$

where

$$A = \mu_0 C_{ox} \frac{W_{eff}}{L_{eff}} V_T^2 e^{1.8} \tag{2-4}$$

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 Figure 2-5. 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,

$$I_{leak} = I_0 e^{\frac{V_{dd}}{nV_T}} \tag{2-5}$$

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

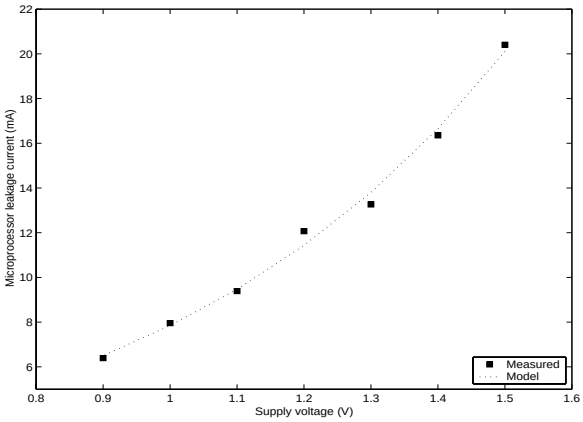


Figure 2-5: Leakage current variation

2.4.1 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 2-6. Equation 2-3 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 2-3 is almost 1. It has been shown in [23] 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 2-3 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 [23] that the ratio of the leakage currents for the three cases shown in Figure 2-6 can be written as

$$I_{l1} : I_{l2} : I_{l3} = 1.8e^{\frac{\eta V_{DD}}{nV_T}} : 1.8 : 1 \qquad (2\text{-}6)$$

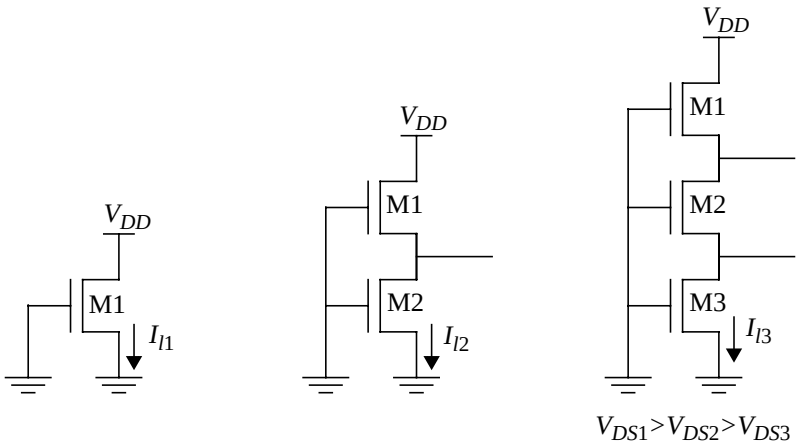


Figure 2-6: Effect of transistor stacking

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 2-6). If the number of stacked devices is more than three, the leakage current contribution from that portion of the circuit is negligible if all transistors are 'OFF'. 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.

2.4.2 Separation of Current Components

Table 2-1 compares the measured leakage current with the values predicted by Equation 2-5. The maximum percentage error measured was less than 6% over the entire operating voltage range of the StrongARM which suggests a fairly robust model. Based on the leakage model described by Equation 2-5, the static and dynamic components of the microprocessor current consumption were separated. These currents have been plotted in Figure 2-7. 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.

Table 2-1: 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

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

$$\beta_{leak} = \frac{I_0 e^{\frac{V_{dd}}{nV_T}}}{I_0 e^{\frac{V_{dd}}{nV_T}} + kV_{dd}^{\alpha}} \qquad (2\text{-}7)$$

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 differentiate Equation 2-7 and solve for extrema. For the StrongARM, this is about 1.2 V as shown in Figure 2-8. The fact that such a minima occurs in measured current profiles further validates our model.

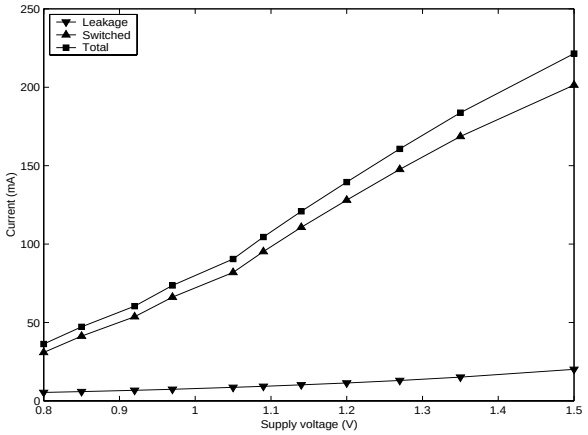


Figure 2-7: Static, dynamic and total current

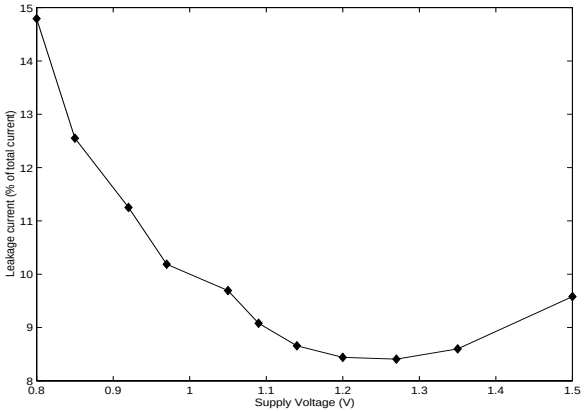


Figure 2-8: Leakage current fraction

2.5 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 2-2. The total switched capacitance and be obtained by dividing out the supply voltage factor from the switching energy.

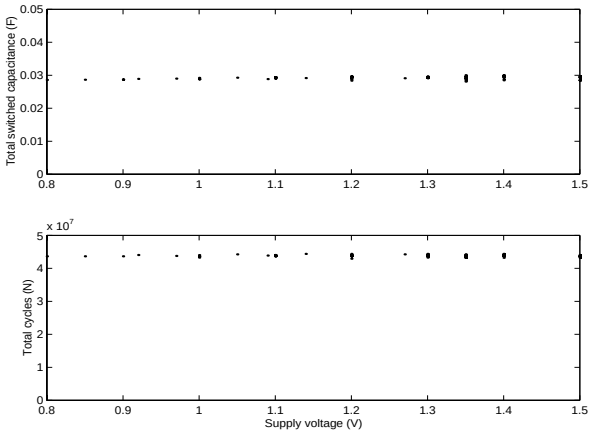


Figure 2-9: Switched capacitance and total cycles

Figure 2-9 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. 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 average total 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.

Figure 2-10 compares the experimental energy data with the modeled values using the total switched capacitance of 0.0292 F and the leakage model of Section 4. The error is less than 1% for the two frequencies shown.

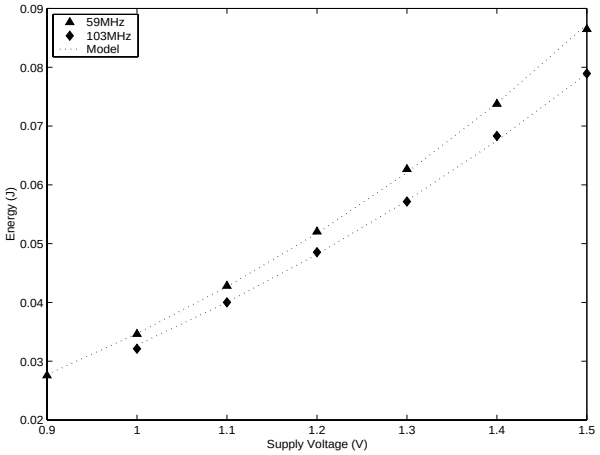


Figure 2-10: Experimental vs modeled energy

2.6 Energy Trade-Offs

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 Figure 2-11. 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.

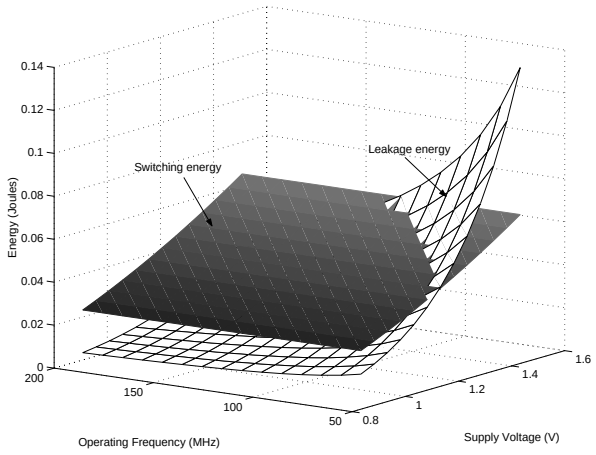
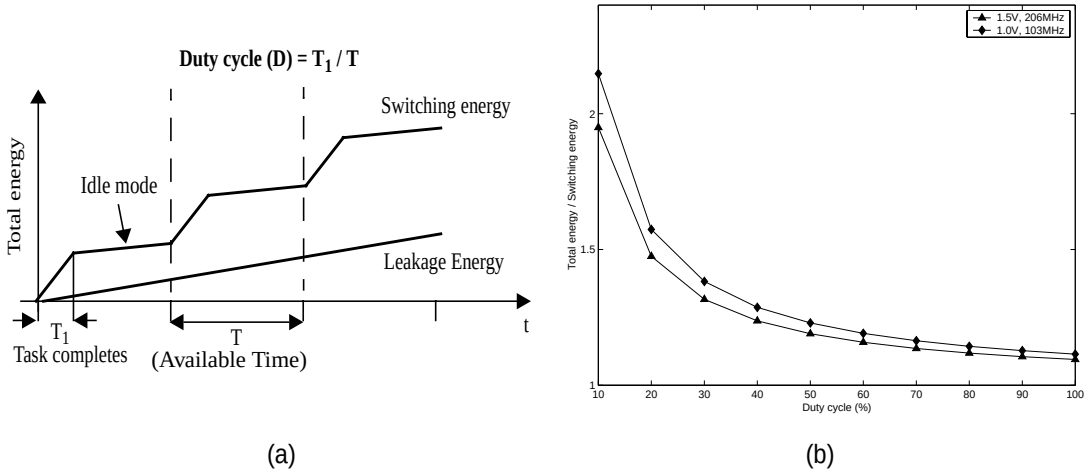


Figure 2-11: FFT energy components

Figure 2-12 illustrates the effect of duty cycle on the energy consumption of a system.



(a) (b)

Figure 2-12: Low duty cycle effects

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. Figure 2-12 (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.

2.7 Results

We propose the following parameterized model for software. The model has been verified on the StrongARM SA-1100 microprocessor,

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

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 defined in Section 2.4, 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. However, the pure exponential behavior of leakage currents might not be completely valid if the processor is dominated by static bias currents (such as from PLLs).

Table 2-2: Model performance

Program	Measured Energy (mJ)	Model			Maximum Error (%)
		C_{tot} (mF)	N (x10^6)	$\frac{C_{tot}}{N}$	
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

Table 2-2 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 2-2 is close to 0.67 nF for all the programs.

2.8 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 and shows less than 5% prediction error for a set of benchmark programs. 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.

Chapter 3
Energy Scalable Algorithms

3.1 Motivation

It is highly desirable that we structure our algorithms and systems in such a fashion that computational accuracy can be traded off with energy requirement. At the heart of such transformations lies the concept of *incremental refinement* [8]. Consider the scenario where an individual is using his laptop for a full-motion video telephone application. Based on the current battery state, overall power consumption model [24] and estimated duration of the call, the system should be able to predict its uptime. If the battery life is insufficient, the user might choose to trade-off some quality/performance and extend the battery life of his laptop.

As another example, consider the distributed sensor network scenario [11] being used to monitor seismic activity from a remote basestation. Once again the sensor nodes are energy constrained and have a finite lifetime. It would be highly desirable to have energy scalable algorithms and protocols running on the sensor network. The remote basestation should have the capability to dynamically reduce energy consumption (to prolong mission lifetime if uninteresting events have occurred) by altering the throughput and computation accuracy. This type of behavior necessitates algorithmic restructuring so that every computational step leads us incrementally closer to the output.

A large class of algorithms, as they stand, do not render themselves to such Energy-Quality (*E*-*Q*) scaling. Using simple modifications, the *E*-*Q* behavior of the algorithm can be modified such that if the available computational energy is reduced, the proportional hit in quality is minimal. However, one must ensure that the energy overhead attributed to the transform is insignificant compared to the total energy consumption. It may be possible to do a significant amount of preprocessing such that the *E*-*Q* behavior is close to perfect but we might end up with a situation where the overall energy consumption is higher compared to the unscalable system. This defeats the basic idea behind having a scalable system viz. overall energy efficiency.

In this Chapter, we first introduce the notion of *E*-*Q* scalability and what the ideal *E*-*Q* behavior of an algorithm should be. Subsequently, the *E*-*Q* behavior of two different commonly used signal processing algorithms is illustrated and contrasted to the *E*-*Q* behavior of the transformed algorithm. All energy benchmarking has been done on the StrongARM SA-1100 microprocessor [17][18].

3.2 Energy Scalable Algorithms in Software

We now formalize the notion of a desirable *E*-*Q* behavior of a system. The *E*-*Q* graph of an algorithm is the function $Q(E)$, representing some quality metric (e.q. mean-square error, peak signal-to-noise ratio etc.) as a function of the computational energy $0 \le E \le E_{max}$. There may exist situations where the notion of a quality metric is unclear (e.g. one can argue that a sort algorithm does not have any approximate notion). However, in this work, we are dealing with signal processing algorithms where the notion of a quality metric is unambiguous. Consider two algorithms (I and II) that perform the same function. Ideally, from an energy perspective, II would be a more efficient scalable algorithm compared to I if

$$Q_{II}(E) > Q_I(E) \quad \forall E \tag{3-1}$$

In most practical cases, Equation 3-1 will not hold over all energy values. As shown in Table 3-1, there might be a preprocessing overhead as a result of which the maximum energy consumptions might be different for the two cases (i.e. $E_{max,\,II} > E_{max,\,I}$). Nevertheless, as long as the Equation 3-1 holds over a significant range of computational energies, overall efficiency is assured.

Let us assume that there exists a quality distribution $p_Q(x)$, i.e. from system statistics we are able to conclude that the probability that we would want a quality x is $p_Q(x)$. A typical quality distribution is shown in Figure 3-1. The average energy consumption per output sample can then be expressed as

$$\bar{E} = \int_Q p_Q(x)E(x)dx \tag{3-2}$$

where *E(Q)* is the inverse of *Q(E)*. When the quality distribution is unknown, we would like the *E*-*Q* behavior to be maximally concave downwards (with respect to the energy axis), i.e.

$$\frac{\partial^2 Q(E)}{\partial E^2} \leq 0 \tag{3-3}$$

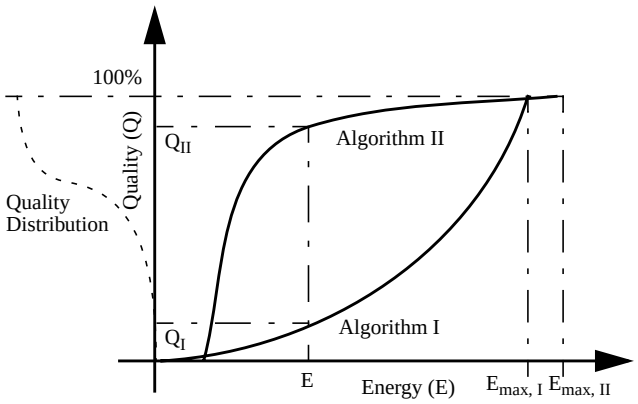


Figure 3-1: *E*-*Q* formal notions

The *E*-*Q* behavior suggested by Equation 3-3 is not always attainable globally i.e. across $0 \leq E \leq E_{max}$ as we will see subsequently. However, on an average case, for a given energy availability *E*, we would like the obtainable quality *Q(E)* to be as high as possible.

Consider the simple power series shown in Equation 3-4. Such power series are frequently encountered in Taylor expansions used to evaluate transcendental functions.

$$y = f(x) = 1 + k_1 x + k_2 x^2 + \ldots + k_N x^N \tag{3-4}$$

A standard implementation of the algorithm would have an *N*-step loop that would multiply the current value of the computed power of *x* with *x* and accumulate the result in *y*. Let us assume we have to compute *f*(2) for *N*=100. If the k_i's are similar, even after *N*-1 steps in the loop, the value accumulated in y would be approximately 50% off from the final value since $2^N / f(2) \approx 1/2$. In terms of *E*-*Q* performance, the algorithm does not do well. Assuming that the amount of energy required to evaluate *f*(2) on a processor is E_{max},

and that each step dissipates the same amount of energy (ignoring inter-instruction effects etc.), we have about 50% computational accuracy after dissipating $(N\text{-}1)/N.E_{max}$ energy. However, if we had to evaluate f(0.5), the most significant terms would occur in the first few steps in the loop and the *E-Q* behavior would be better. Based on the above analysis, we can conclude that transforming the algorithm, as shown in Table 3-1, will result in the most significant computations occurring early in the loop as a result of which, the computational energy could be reduced, without taking a significant hit in accuracy.

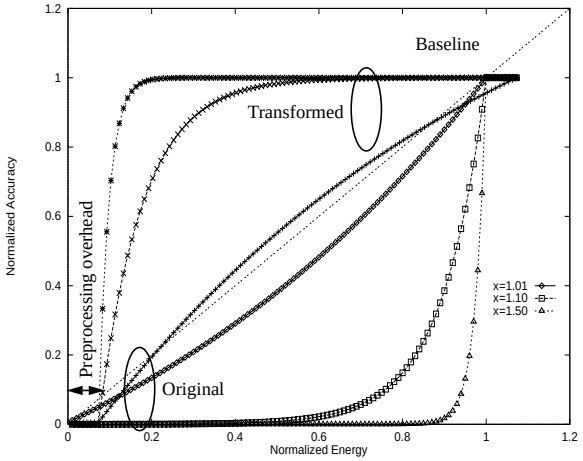


Figure 3-2: *E-Q* performance of power series algorithm

Table 3-1: Power series computation

Original Algorithm	Transformed Algorithm

`xpowi = 0.0; y = 1.0;`
`for( i=1; i<N; i++ ) {`
`  xpowi *= x*k[i];`
`  y += xpowi;`
`}`
	`if( x>1.0 ) {`
	`  xpowi = k[N]*pow(x,N);`
	`  y = xpowi+1;`
	`  for( i=N-1; i>0; i-- ) {`
	`  xpowi /= x*k[i];`
	`  y += xpowi; }`
	`}`
	`else { // original algorithm`
	`}`

Figure 3-2 shows the *E-Q* graphs for the original and modified power series algorithm. It captures the all the basic ideas. (i) *E-Q* behavior is in general data dependent. It is possible to come up with pathological cases where the transformed algorithm would have a *E-Q* behavior very close to the original. However, from an energy efficiency perspective, its the average *E-Q* performance that matters. (ii) It is desirable to have an *E-Q* graph above the

baseline ($E=Q$ on a normalized scale). This would imply that marginal returns in accuracy from successive units of computational energy is diminishing. Therefore, if the available energy is reduced by 10%, the quality degradation is less that 10%, the lesser, the better. (iii) There is an energy overhead associated with the transform which should be insignificant compared to the total energy.

3.3 Energy Scalable Transformations

3.3.1 Filtering Application

Finite Impulse Response (FIR) filtering is one of the most commonly used Digital Signal Processing (DSP) operations. FIR filtering involves the inner product of two vectors one of which is fixed and known as the impulse response, $h[n]$, of the filter [20]. An N-tap FIR filter is defined by Equation 3-5.

$$y[n] = \sum_{k=0}^{N-1} x[n-k]h[k] \tag{3-5}$$

Various low power and energy efficient implementations of the FIR filter have been proposed and implemented. The approximate processing techniques proposed in [25] reduce the total switched capacitance by dynamically varying the filter order based on signal statistics.

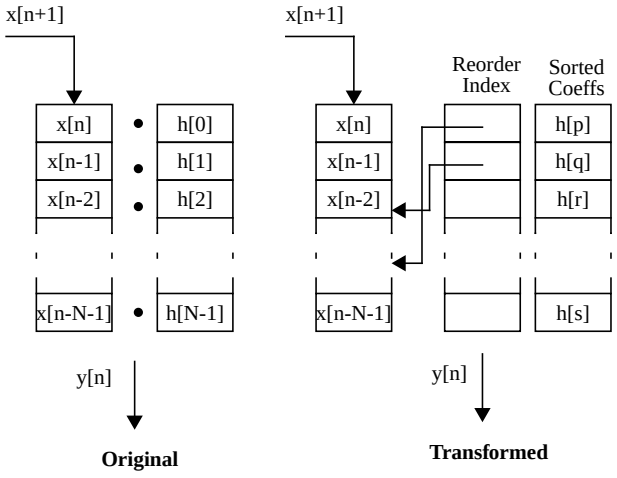


Figure 3-3: FIR filtering with coefficient

However, when we analyze the FIR filtering operation from a pure inner product perspective, it simply involves N multiply and accumulate (MAC) cycles. For desired E-Q behavior, the MAC cycles that contribute most significantly to the output $y[n]$ should be done first. Each of the partial sums, $x[k]h[n-k]$, depends on the data sample and therefore its not apparent which ones should be accumulated first. Intuitively, the partial sums that are maximum in magnitude (and can therefore affect the final result significantly) should be accumulated first. Most FIR filter coefficients have a few coefficients that are large in magnitude and progressively reduce in amplitude. Therefore, a simple but effective *most-significant-first transform* involves sorting the impulse response in decreasing order of magnitude and reordering the MACs such that the partial sum corresponding to the largest coefficient is accumulated first as shown in Figure 3-3. Undoubtedly, the data sample multiplied to the coefficient might be so small as to mitigate the effect of the partial sum. Nevertheless, on an average case, the coefficient reordering by magnitude yields a better E-Q performance than the original scheme especially since correctly sampled data will have some amount of sample to sample correlation. Figure 3-4 illustrates the scalability results for a low pass filtering of speech data sampled at 10kHz using a 128-tap FIR filter whose impulse response (magnitude) is also outlined. The accuracy metric is the mean square error from the final result. The average energy consumption per output sample (measured on the StrongARM SA-1100 operating at 1.5V power supply and 206MHz frequency) in the original scheme is 5.12μJ. Since the initial coefficients are not the ones with most significant magnitudes the E-Q behavior is poor. Sorting the coefficients and using a level of indirection (in software that amounts to having an index array of the same size as the coefficient array), the E-Q behavior can be substantially improved. It can be seen that fluctuations in data can lead to deviations from the ideal behavior suggested by Equation 3-3, nonetheless overall concavity is still apparent. The energy overhead associated with using a level of indirection on the SA-1100 was only 0.21μJ which is about 4% of the total energy consumption.

In FIR filtering, the input data samples are unknown a priori. The partial sum which is most significant is not completely deterministic until all of them have been computed. More sophisticated schemes could involve sorting both the data samples and the coefficients and using two levels of indirection to perform the correct inner product first by pick-

ing up the partial sum corresponding to the largest coefficient, then the one corresponding to the largest data sample and so on. The overhead associated with such a scheme involves real time sorting of incoming samples. Assuming that we have a presorted data array at time *n*, the next data sample $x[n+1]$ can be inserted into the right position using a binary search type technique which can be done in $O(logN)$. The scalability gains might not be substantial compared to the simpler scheme discussed before. However, in applications such as autocorrelation which involves an inner product of a data stream with a shifted version of itself, sorting both the vectors in the inner product would yield significant improvements in *E-Q* behavior.

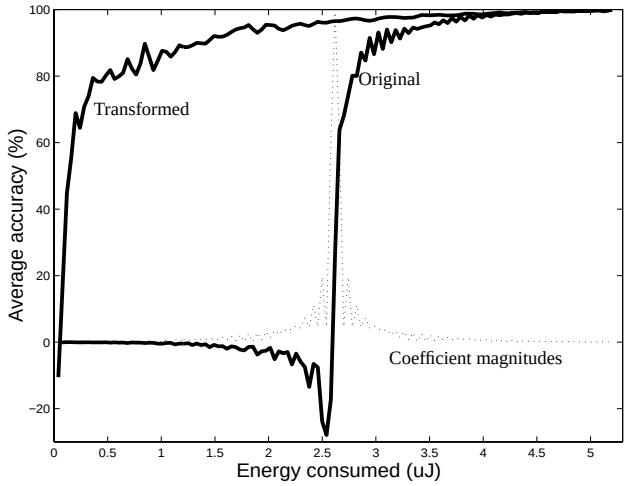


Figure 3-4: *E-Q* graph for original and

3.3.2 Image Decoding Application

The Discrete Cosine Transform (DCT), which involves decomposing a set of image samples into a scaled set of discrete cosine basis functions, and the Inverse Discrete Cosine Transform (IDCT), which involves reconstructing the samples from the basis functions, are crucial steps in digital video [26]. The 64-point, 2-D DCT and IDCT (used on 8x8 pixel blocks in of an image) are defined respectively as

$$X[u,v] = \frac{c[u]c[v]}{4}\sum_{i=0}^{7}\sum_{j=0}^{7}x[i,j]\cos\left(\frac{(2i+1)u\pi}{16}\right)\cos\left(\frac{(2j+1)v\pi}{16}\right) \qquad (3\text{-}6)$$

$$x[i,\ j]\ =\ \frac{1}{4}\sum_{u\ =\ 0}^{7}\sum_{v\ =\ 0}^{7}c[u]c[v]X[u,\ v]\cos\left(\frac{(2i+1)u\pi}{16}\right)\cos\left(\frac{(2j+1)v\pi}{16}\right) \qquad (3\text{-}7)$$

DCT is able to capture the spatial redundancy present in an image and the coefficients obtained are quantized and compressed. Most existing algorithms attempt to minimize the number of arithmetic operations (multiplications and additions) usually relying on the symmetry properties of the cosine basis functions (similar to the FFT algorithm) and on matrix factorizations [27]. The *E-Q* behavior of these algorithms are not good as they have been designed such that computation takes a minimal yet constant number of operations. The Forward Mapping-IDCT (FM-IDCT) algorithm, proposed in [9] can be shown to have an *E-Q* performance with is much better than other algorithms. The algorithm is formulated as follows

$$\begin{bmatrix} x_{0,\ 0} \\ x_{0,\ 1} \\ \vdots \\ x_{8,\ 8} \end{bmatrix}\ =\ X_{0,\ 0}\begin{bmatrix} c_0^{0,\ 0} \\ c_1^{0,\ 0} \\ \vdots \\ c_{64}^{0,\ 0} \end{bmatrix} + X_{0,\ 1}\begin{bmatrix} c_0^{0,\ 1} \\ c_1^{0,\ 1} \\ \vdots \\ c_{64}^{0,\ 1} \end{bmatrix} + \ldots + X_{8,\ 8}\begin{bmatrix} c_0^{8,\ 8} \\ c_0^{8,\ 8} \\ \vdots \\ c_{64}^{8,\ 8} \end{bmatrix} \qquad (3\text{-}8)$$

where $x_{i,\ j}$ are the reconstructed pels, $X_{i,\ j}$ are the input DCT coefficients, and$[c_k^{i,\ j}]$ is the 64x64 constant reconstruction kernel.

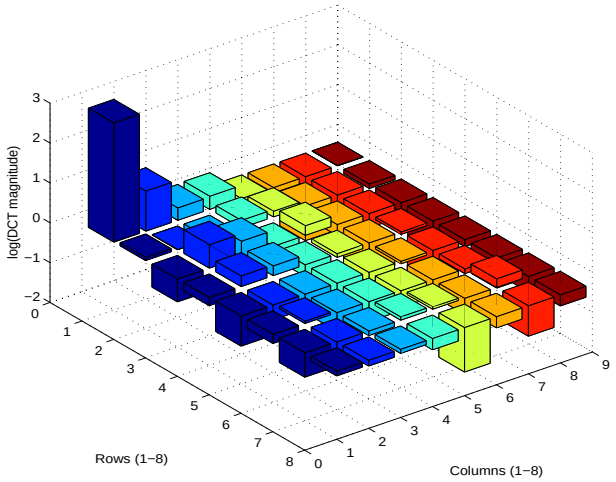


Figure 3-5: 8x8 DCT coefficient magnitudes

The improved *E-Q* behavior of the FM-IDCT algorithm can be attributed to the fact that most of the signal energy is concentrated in the dc coefficient ($X_{0,0}$) and in general in the low-frequency coefficients as shown in Figure 3-5. Instead of reconstructing each pixel by summing up all its frequency contributions, the algorithm incrementally accumulates the entire image based on spectral contributions from the low to high frequencies.

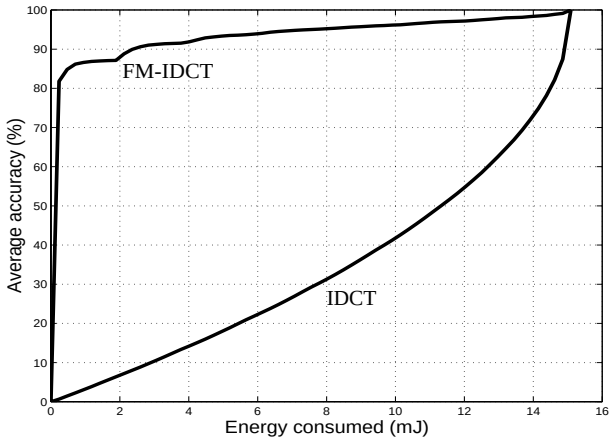


Figure 3-6: *E-Q* graph for FM-IDCT vs normal IDCT

Figure 3-6 and Figure 3-7 illustrate the *E-Q* behavior of the FM-IDCT algorithm. It is obvious from Figure 3-7 that almost 90% image quality can be obtained from as little as 25% of the total energy consumption. In terms of the overhead requirement, the only change that is required is that we now need to store the IDCT coefficients in a transposed fashion (i.e. all the low frequency components first and so on).

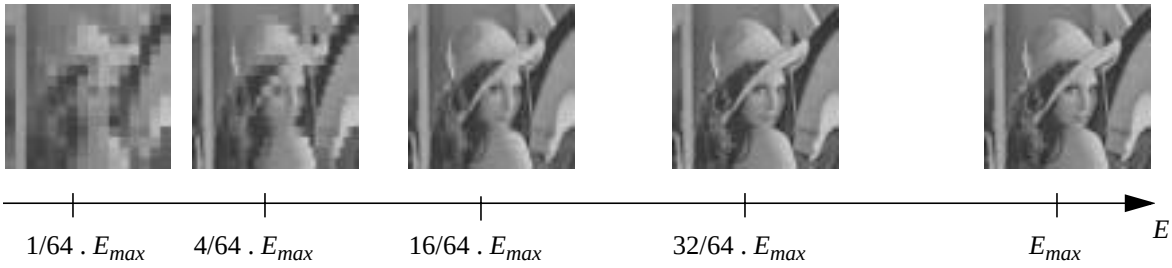


Figure 3-7: Incremental refinement property of the FM-IDCT algorithm

Java Based Real-Time Energy Scalable Video Application

The Scalable Video program was developed in Java (version 1.2.2) to test the ideas discussed above. The most important Java classes are shown in Figure 3-8.

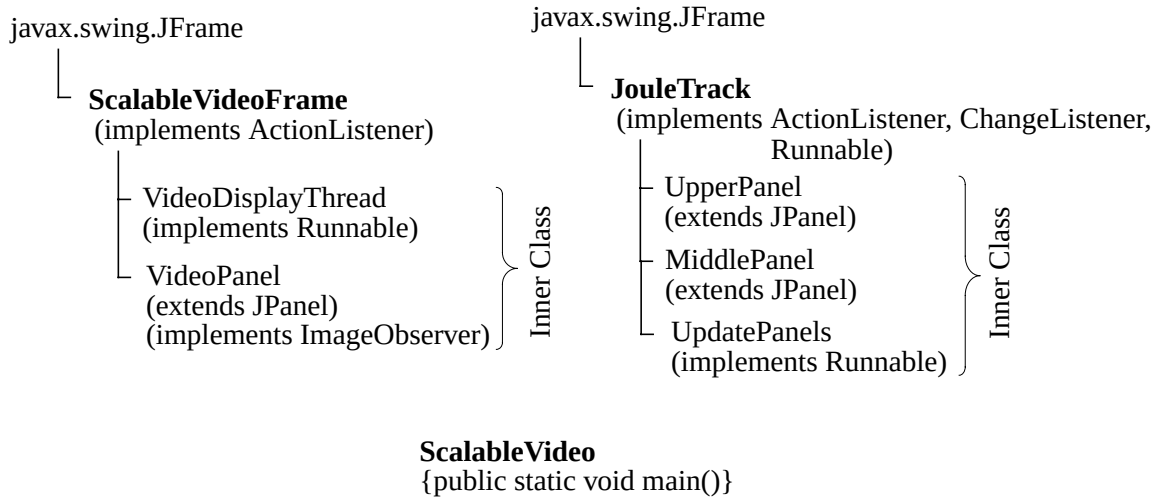


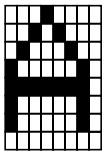
Figure 3-8: Main classes and their hierarchy

The class **java.awt.image.MemoryImageSource** [28] was used to produce images from the RGB pixel values computed after the IDCT step. This class is an implementation of the ImageProducer interface which uses an array to produce pixel values for an Image. The following code snippet shows how the images were created and displayed:

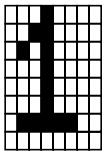
```
   while( current == videoOn ) {
      try {
        redFile = new FileInputStream( filename+".red" );
        greenFile = new FileInputStream( filename+".green" );
        blueFile = new FileInputStream( filename+".blue");
        redFile.read( red );
        blueFile.read( blue );
        greenFile.read( green ); // the RGB values are in separate files.
   // Each pixel being represented by a byte value between 0-255.
        for( i=0; i<size; i++ ) {
          tempr = (red[i]<<16) & rmask;
          tempg = (green[i]<<8) & gmask;
          tempb = blue[i] & bmask;
          pixels[i] = (alpha | tempr | tempg | tempb);//Create the pixel
        }
         videoFrame = createImage( new MemoryImageSource(width, height,
pixels, 0, width) );
        redFile.close();
        blueFile.close();
        greenFile.close();

        repaint(1);
```

```
        currFrameNo = (currFrameNo+1) % MAX_FRAMES;
      }
      catch( FileNotFoundException e1 ) {
        System.out.println( "\n Video Frame not found : " + filename );
      }
      catch( IOException e2 ) {
        System.out.println( "\n File contains invalid data : " + filename
);
      }
   }
```

The JouleTrack class displays the processor energy, voltage, current and frequency. It also has a slider that lets the user set the accuracy level. To mimic the output of a SourceMeter (like a digital ammeter, voltmeter etc.) we created a special LED display. Each font was stored in a hashtable, using an 8x8 pixel representation typically used by LEDs. The characters were first generated using the "banner" utility in UNIX and then a hashtable was created with each character represented by a string of 64 boolean values representing the 8x8 character display as shown in Figure 3-9.



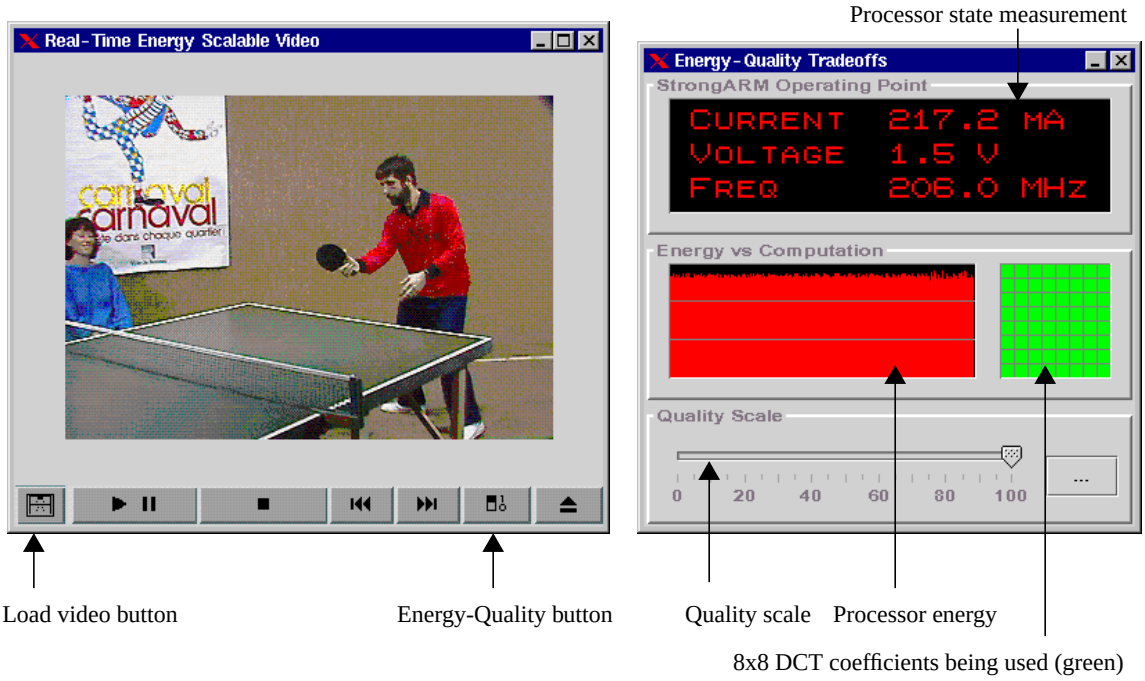
'A' = 0001000000101000010001001000001011111110100000101000001000000000



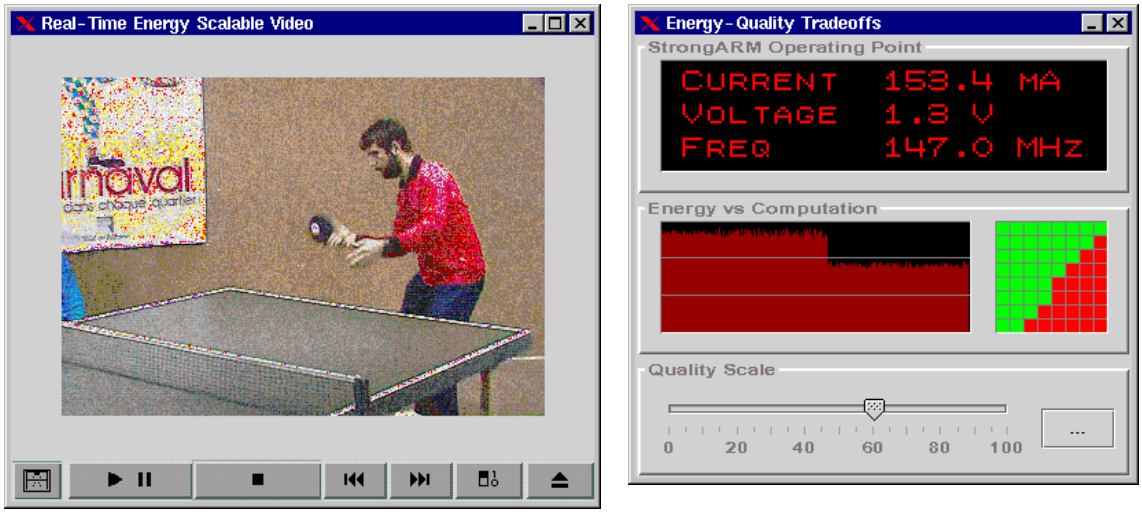
'1' = 0001000000110000010100000001000000010000000100000111110000000000

Figure 3-9: LED display hashtable entries (each pixel is represented by a 1 or 0)

The JouleTrack class also displays the IDCT coefficients being used by the ScalableVideoFrame class. At 100% computational accuracy, all the coefficients are used for computation. At reduced accuracy, only the most significant ones (i.e. the lower frequency ones) are used to create the image. A few sample screenshots from the application is are shown in Figure 3-10.



(a) Video at 100% accuracy, 1.5V, 206MHz



(b) Video at 60% accuracy, 1.3V, 147MHz

Figure 3-10: Screenshots from the software

3.4 Conclusions

We have introduced the notion of energy scalable computation in the context of signal processing. Algorithms that render incremental refinement of a certain quality metric such that the marginal returns from every additional unit of energy is diminishing are highly desirable in embedded applications. Using three broad classes of signal processing algorithms we have demonstrated that using simple transformations (with insignificant overhead) the Energy-Quality (*E*-*Q*) behavior of the algorithm can be significantly improved. In general, we have concluded that doing the most significant computations first enables computational energy reduction without significant hit in output quality.

Chapter 4
Energy Scalability in Hardware

4.1 Motivation

To attain system energy scalability, all components viz. algorithms to hardware must be made scalable. For instance, running an FIR filtering application on a fixed precision DSP can give us only a limited amount of scalability as discussed in Chapter 3. However, if we look at a typical data stream (e.g. speech samples being low-pass filtered) the amount of precision required to perform the operation is variable. Most contemporary designs take into account the worst case data precision requirement while designing the datapath. Therefore, even though the immediate precision requirement might be less, lots of energy is wasted in running the computations with the worst case datapath precision. In this chapter we introduce the concept of *precision-on-demand* and show how additional energy scalability (without sacrificing any further quality) can be obtained by having a scalable datapath using the FIR filtering operation as an example.

There exists a wealth of techniques proposed in literature for low power implementations of FIR filters. When the filter coefficients are fixed, the flexibility offered by a multiplier is not required. Distributed Arithmetic (DA) is a bit-serial, multiplier-less technique that exploits the fact that one of the vectors in the inner product is fixed [31]. All possible intermediate computations (for the fixed vector) are stored in a Lookup Table (LUT) and bit slices of the variable vector are use as addresses for the LUT. A 4-tap DA based FIR filter is shown in Figure 2-1. In general, an N tap filter requires an LUT of size 2^N. Area delay trade-offs have been explored in [32]. For instance, the LUT of size 2^N can be split up into two different LUTs of size $2^{N/2}$ at the cost of an additional adder.

The output $y[n]$ is not available every cycle. If the bit precision of the data samples $x[n\text{-}k]$ is M, then a valid output $y[n]$ is available every M cycles. In the MSB first implementation of Figure 2-1, it has been shown in [33][34] that each successive intermediate value is closer to the final value in a stochastic sense.

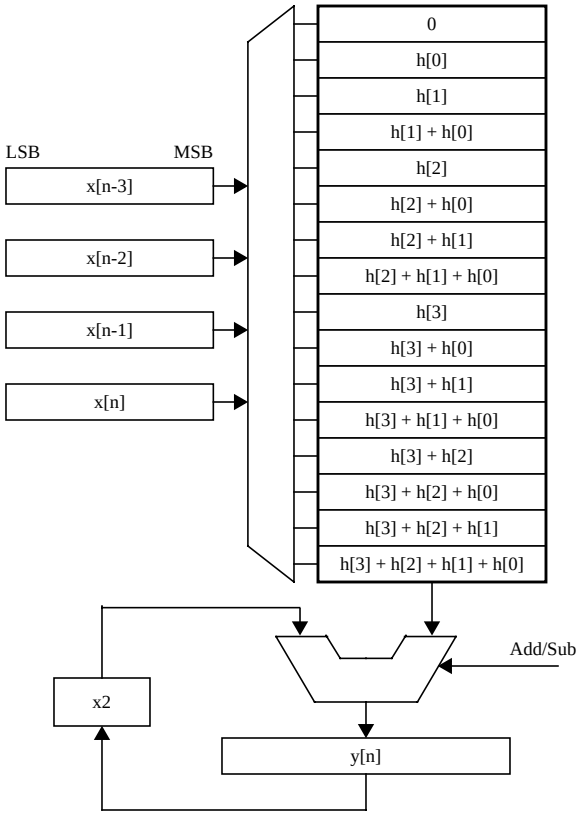


Figure 4-1: DA implementation of a four tap filter

If we assume that the energy consumption per cycle is E_0, it is straightforward to see that the energy consumption per output sample is ME_0, i.e. the energy consumption is linearly dependent on the bit precision of data samples. Of course, we have to design the filter so that it can accommodate the largest possible input signals (i.e. those requiring the maximum possible bits for representation). In most signal processing applications, the signals we get are correlated and only few samples actually require the maximum bit precision for representation. Figure 4-2 shows 4 seconds of speech data sampled at 10kHz. For most of the 40,000 samples we see that the bit precision required for representation is significantly less that 16 bits. In fact, the average precision required is about 6.9 bits. Most filtering circuits will be designed to accommodate 16 bit data, and in our DA based implementation, with fixed precision, the energy required per output sample would be $16E_0$.

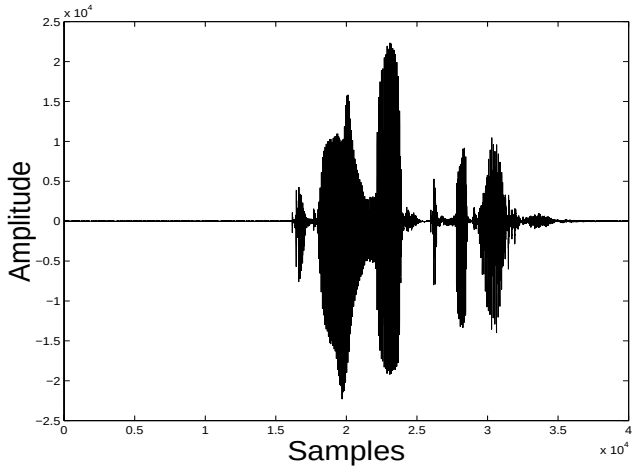


Figure 4-2: Speech data sampled at 10kHz, 16 bit precision

Our variable bit precision filtering scheme that saves energy in two ways, without loss in any accuracy. First, we use only that amount of precision as immediately required by the samples. Let us assume that the maximum precision requirement is M_{max} and the immediate precision requirement is $M \leq M_{max}$.This scales down the energy per output sample by a factor M/M_{max}. Further, we exploit the fact that lesser precision implies that the same computation can be done faster (i.e. in M cycles instead of M_{max}). We therefore switch down the operating voltage such that we still meet the worst case throughput requirement (i.e. corresponding to one output sample every M_{max} cycles when operating at V_{max}) while obtaining quadratic energy savings.

In this chapter, we present a novel Finite Impulse Response (FIR) filter architecture based on a Distributed Arithmetic (DA) approach with two supply voltages and variable bit precision operation. The filter is able to adapt itself to the minimum bit precision required by the incoming data and also operate at a lower voltage so that it still meets a fixed throughput constraint. As opposed to the worst case fixed precision design, our precision-on-demand implementation has an energy requirement that varies linearly with the average bit precision required by the input signal. We also demonstrate that 50% to 60% energy savings can easily be obtained in the case of speech data.

4.2 Filter Architecture

DA is particularly suited to variable precision filtering. The filter that we have implemented is an 8-tap low pass filter. Both data and coefficients use a 16 bit two's complement representation. The LUT has 2^8 i.e. 256 entries. There are 8 data registers (corresponding to the 8 data samples that are required for every output sample).

4.2.1 Determining Precision Requirement

To implement a precision-on-demand scheme, we need to determine the minimum precision that is required to compute the output without any loss in accuracy. Consider the two cases shown in Figure 4-3 (a) and (b).

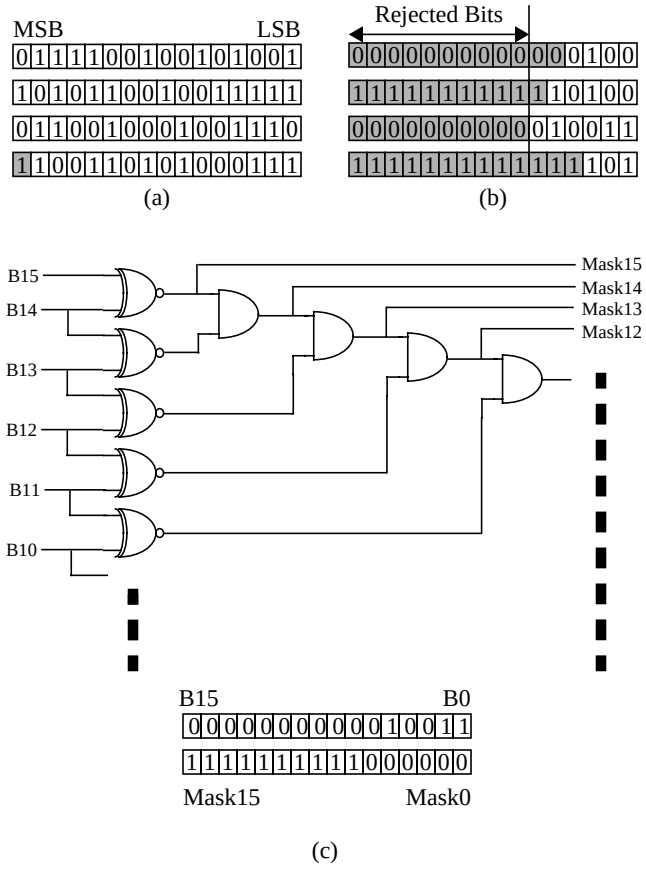


Figure 4-3: Determining sign extension bits

In both cases (a) and (b) we have 4 registers, each with two's complement 16 bit data. The sign extension bits have been shaded in every register. Notice that in case of small

numbers the MSBs are ‘0’ for positive quantities and ‘1’ for negative quantities. No accuracy is lost if we reject the sign extension bits. Determining the number of sign extension bits is relatively simple in hardware and can be done by the circuit shown in Figure 4-3 (c) [33]. The ‘1’ outputs of the ‘mask’ determine the sign extension bits in each register. The number of sign extension bits that can be rejected is equal to the minimum of the sign extension bits among all the registers. This can be obtained by simply ‘anding’ all the individual mask outputs from each register. In our DA implementation, the final mask (obtained by ‘anding’ all the individual masks) determines the number of cycles, M, required for computing the current output. For a 4-tap implementation, with the register contents as shown in Figure 4-3 (b), the number of cycles that will be required is $M = 6$, instead of $M_{max} = 16$. The energy overhead due to the sign extension hardware is small since the duty cycle is low (one per output sample).

4.2.2 Just-In-Time Computation

Once the precision requirement for a sample has been determined we know the number of cycles, M, required for obtaining the result $y[n]$. In general, the filter will have to be designed such that it meets a fixed throughput requirement in the worst case operating precision, i.e. we have one output sample $y[n]$ at least every M_{max} cycles. But with a precision requirement $M \leq M_{max}$, we would have a valid output earlier. Since, it does not pay to do computations any faster than required, we would have to idle for (M_{max} - M) cycles. We can exploit this fact, that we have more time for computation, by lowering the supply voltage. This results in quadratic reduction in energy and corresponding increase in computational delay.

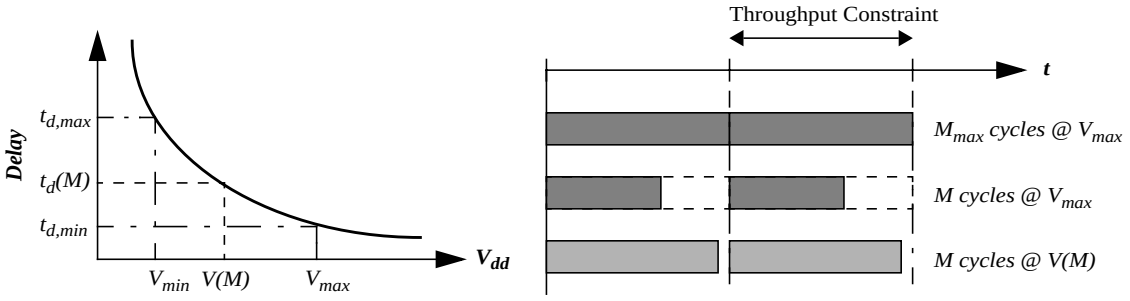


Figure 4-4: Just-in-time computation

We lower the supply voltage, V_{dd}, from $V_{dd} = V_{max}$, to $V_{dd} = V(M)$, such that the time required to execute M cycles at $V_{dd} = V(M)$ is approximately equal (but not greater than) the time required to execute M_{max} cycles at $V_{dd} = V_{max}$. This idea is summarized in Figure 4-4.

Implementing a scheme like this involves a variable supply voltage [10] (a DC/DC converter) with sufficient feedback bandwidth, a variable clock frequency generator along with the precision determining circuit in the original DA implementation as shown in Figure 4-5. Even the best DC/DC converters do not have sufficient feedback bandwidth to track the fast changes in data precision requirement that filtering involves. Therefore such a scheme is difficult to implement practically.

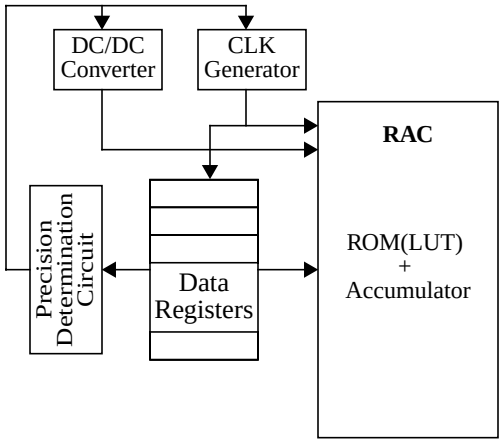


Figure 4-5: Basic feedback structure for just-in-time computation

4.2.3 Practical Implementation with Two Supply Voltages

To overcome the limitation posed by the DC/DC converter we implemented and tested a filtering scheme that uses two RAC units operating at two different supply voltages, $V_{dd} = V_{max}$ and $V_{dd} = V_{min.}$ The RAC operating at the higher voltage is such that it can produce a valid output every M_{max} cycles and meets the throughput constraint. The RAC operating at the lower voltage is such that it can go through only M_{min} cycles before it exceeds the throughput constraint. If the current precision requirement is M ($M_{min} \le M \le M_{max}$), we execute some fraction, f, of M_{max} cycles, at the higher voltage RAC, and the remaining

cycles, $(1 - f)$, of M_{min} cycles, at the lower voltage RAC. Of course, the number of cycles we spend at either voltages, must be integral and must add up to *M*.

$$f \cdot M_{max} + (1 - f) \cdot M_{min} = M \tag{4-1}$$

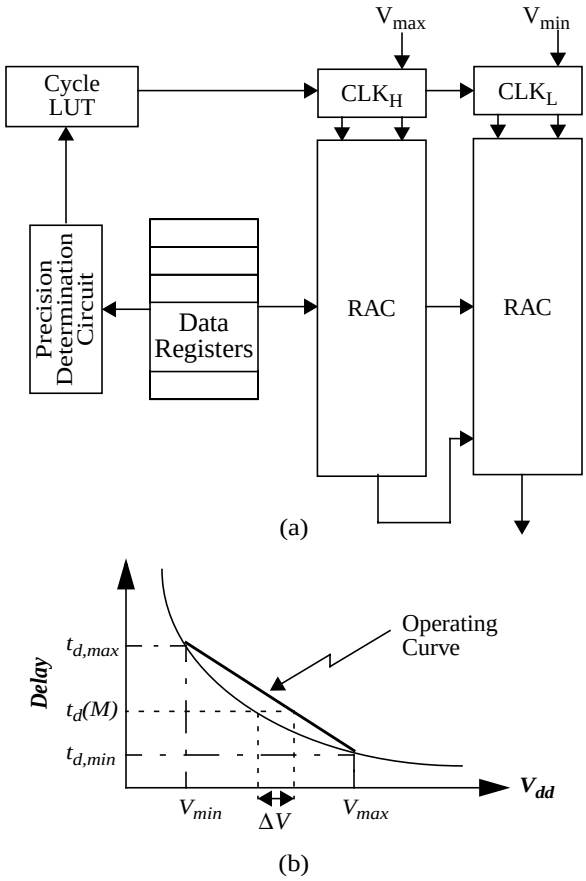


Figure 4-6: Just-in-time filtering with two supply voltages

Figure 4-6 illustrates the basic architecture of just-in-time filtering with two RAC units operating at two supply voltages. The details of the control logic and level shifters have been omitted for clarity. The operation is very simple. When a new data sample is loaded into the register, the 'precision determination circuit' computes the minimum precision required for the computation. This determines the total number of cycles, *M*. The accumulators within each of the RACs are cleared. The 'cycle LUT' then determines the number of cycles to be executed at V_{max} and V_{min} respectively and clocks the corresponding RACs. At the end of fM_{max} cycles at V_{max}, the accumulator contents of the RAC operat-

ing at V_{max} are loaded into the RAC operating at V_{min}, and the RAC is clocked for $(1-f)M_{min}$ cycles.

The average delay per cycle in this scheme is therefore a simple linear combination of the delays $t_{d,max}$ and $t_{d,min}$. This is suboptimal compared to the arbitrary voltage scheme as shown in Figure 4-6 (b) in that we are operating at an average voltage which is higher than the optimal voltage by ΔV.

In our implementation, M_{min} was chosen to be 4. If the precision requirement was less than 4, all cycles were executed at V_{min}. Therefore, once again, the two voltage scheme was suboptimal in that we did not exploit all the time we had for computation. The choice of M_{min} is data dependent. We do not want it to very small because most of the cycles would then be spent at V_{max}. At the same time we do not want it to be large because all precision requirements below M_{min} would then be suboptimal. In the next section we demonstrate an algorithm to choose the optimal M_{min} (and therefore V_{min}).

4.3 Theoretical Analysis

We begin with the assumption that M_{max} is fixed (determined by the maximum bit precision in the data representation which meets a specified accuracy requirement). Let the throughput requirement be one output sample every T_0 time duration. In our DA based implementation, this translates to a RAC that can execute M_{max} clock cycles within T_0 time i.e. the worst case critical path delay should be less than T_0/M_{max}. Given a particular process technology, and RAC design, we have a delay-voltage characteristic similar to Figure 4-4 which we represent as

$$t_d = g(V_{dd}) \tag{4-2}$$

Therefore, using $t_{d,min} = T_0/M_{max}$ and the delay-voltage characteristic, we can determine V_{max}.

Suppose, we also choose M_{min} (we shall establish an algorithm to optimally determine M_{min}). Once again, using $t_{d,max} = T_0/M_{min}$ and the delay-voltage characteristic, we can determine V_{min}. Let the instantaneous precision requirement be M. We need to determine

the fraction of cycles that we need to execute at each voltage. From Equation 4-1, we compute

$$f = \frac{M - M_{min}}{M_{max} - M_{min}} \tag{4-3}$$

Of course, f should be such that fM_{max} and $(1-f)M_{min}$ are integral. It must also satisfy the throughput constraint

$$fM_{max}t_{d,min} + (1-f)M_{min}t_{d,max} \leq T_0 \tag{4-4}$$

Figure 4-7 illustrates how fM_{max} and $(1-f)M_{min}$ vary for different M. As before, $M_{max} = 16$ and $M_{min} = 4$.

Next we determine the energy consumption per sample as a function of the required precision. Let us assume that the energy dissipated per cycle at $V_{dd} = V_{max}$ is ΔE_0. Clearly, in the fixed precision DA, the energy consumption per cycle is $E_0 = M_{max}\Delta E_0$. If we neglect the energy overhead due to the precision determination circuit and extra controls, the energy as a function of M is

$$E(M) = fM_{max}\Delta E_0 + (1-f)M_{min}\Delta E_0\left(\frac{V_{min}}{V_{max}}\right)^2 \tag{4-5}$$

and substituting Equation 4-3, we get

$$E_R(M) = \alpha M - \beta \tag{4-6}$$

where $E_R(M)$ is the energy normalized with respect to the maximum energy E_0 and the constants are

$$\alpha = \frac{M_{max} - M_{min}\left(\frac{V_{min}}{V_{max}}\right)^2}{M_{max}(M_{max} - M_{min})} \tag{4-7}$$

$$\beta = \frac{M_{max}M_{min}\left(1-\left(\frac{V_{min}}{V_{max}}\right)^2\right)}{M_{max}(M_{max}-M_{min})} \tag{4-8}$$

We can clearly see the linear variation of energy with precision requirement. The voltage ratio determines the slope of the variation and also the offset. Equation 4-6 is valid only for the range $M_{min} \le M \le M_{max}$. As we pointed out before, for $M < M_{min}$, all the cycles are executed at V_{min} (i.e. $f = 0$).

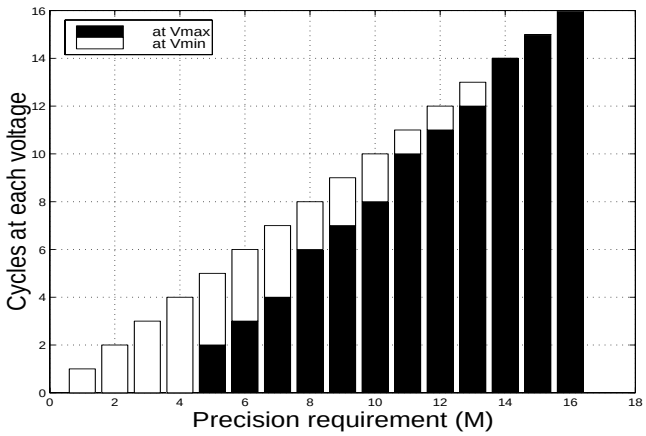


Figure 4-7: Cycle distribution for different precision requirements

Next, we need to determine the optimum operating voltage, V_{min}, and the corresponding precision. This cannot be done without an a priori knowledge of data statistics. We need to know the precision distribution profile for the data i.e. the relative frequency of a particular precision requirement in the filter for the given data. Let us assume that p_M is the probability that the precision requirement in the DA based FIR filter is M. The expected normalized energy per sample required by the filter, for the particular data, would be

$$\bar{E}_R = \sum_M p_M \cdot E_R(M) \tag{4-9}$$

and substituting Equation 4-6 we get

$$\bar{E}_R = \alpha\bar{M} - \beta \tag{4-10}$$

where $\overline{M}$ is the average precision requirement in the filter. To compute the optimum V_{min} we have to set

$$\frac{\partial \overline{E}_R}{\partial V_{min}} = 0 \tag{4-11}$$

And using $t_{d,max} = g(V_{min}) = T_0/M_{min}$ we can solve Equation 4-11 and obtain the optimum V_{min} and M_{min}.

4.4 Results

To obtain the delay voltage characteristics we extrapolated the data available from the 4-tap, 10 bit precision RAC design implemented in 0.6µm technology [33]. The delay voltage characteristic for a 8-tap, 16 bit precision RAC is shown in Figure 4-8. This is a conservative characteristic. However, in our case, only the relative characteristic matters and not the absolute delay numbers.

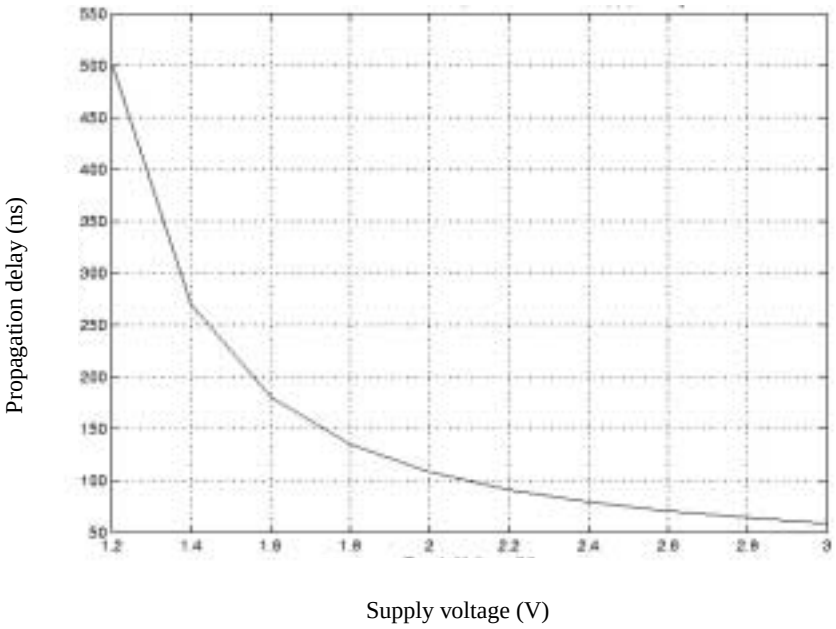


Figure 4-8: Typical delay voltage characteristics of a RAC

In our implementation V_{max} = 2.1V, V_{min} = 1.3V, M_{max} = 16, and M_{min} = 4. Figure 4-9 illustrates the precision distribution for typical speech data (as shown in Figure 4-2). Notice that the distribution peaks around M = 4 which implies that a large fraction of the

data would require only the RAC operating at V_{min} to execute all cycles. Figure 4-10 shows the energy consumption (normalized) sample by sample for the speech data shown in Figure 4-2. It can be clearly seen that the energy requirement tracks the precision requirement. For example, the first 15,000 samples, where the data values are relatively small, the energy requirement is less than 20% of the maximum requirement. On the other hand, the next 10,000 samples being large (requiring close to the maximum precision) use up a lot more energy. The average energy required in this case is about 40% of the maximum i.e. our scheme saves about 60% energy. This is perfect agreement with our previous theoretical analysis.

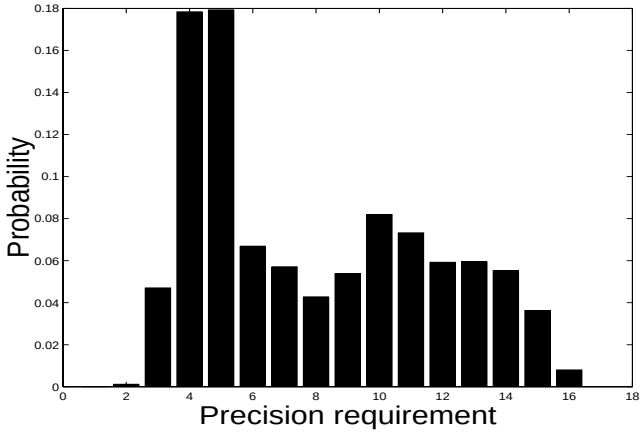


Figure 4-9: Distribution of precision requirement

Figure 4-11 illustrates the time that has to be spent at V_{max} and V_{min} respectively, to meet the throughput requirement at each precision M. Notice that for $M < M_{min}$ (which is 4 in our case), the filter has to idle for some time. Also, all cases $M \geq M_{min}$, are not "just-in-time" because the number of cycles that have to be spent at each voltage is integral.

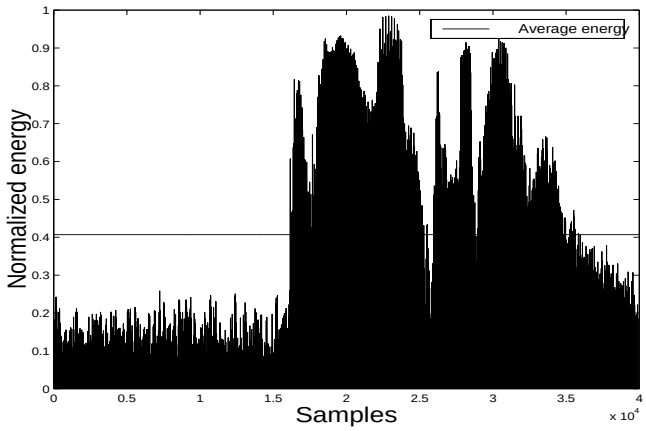


Figure 4-10: Sample by sample energy requirement

Finally, Figure 4-12 shows the results of our filtering scheme on 37 different speech data blocks. We can clearly see the linear variation of average energy requirement with the average precision requirement as was predicted by Equation 4-10. Once again the energy is normalized with respect to the maximum energy. For the fixed precision implementation, all the speech blocks would require almost the same energy equal to the maximum energy.

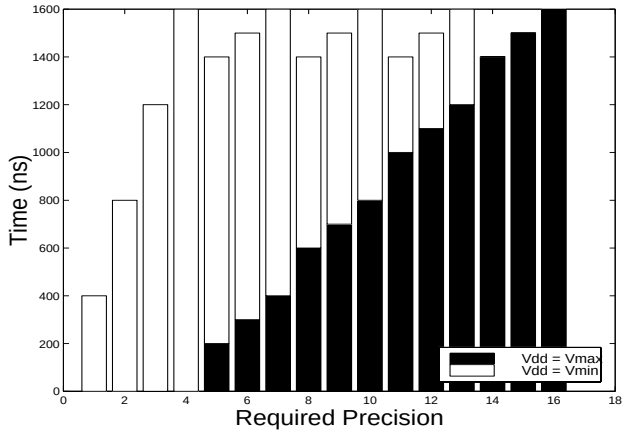


Figure 4-11: Time/cycles spent at each voltage

An interesting case to compare would be the classical area power trade-off that one gets from duplicating hardware [35]. In our case that translates to having two RACs running at some intermediate voltage V_m such that each RAC is able to execute M_{max} cycles in the twice the throughput interval. So, we would have two such units running in parallel

in such a fashion that we have one output sample every T_0 time. Clearly, this would result in an energy reduction by a factor $(V_m/V_{max})^2$. The energy reduction is fixed and independent of precision requirement as shown in Figure 4-12. However, if the data samples are such that the precision requirement is heavy, then the energy advantage of our scheme would be lost. Also plotted in Figure 4-12 is the average energy variation with average precision requirement for the just the variable precision case (i.e. variable precision using single RAC, single operating voltage). The difference in slope and offset, as predicted by Equation 4-7 and Equation 4-8 respectively, is immediately visible.

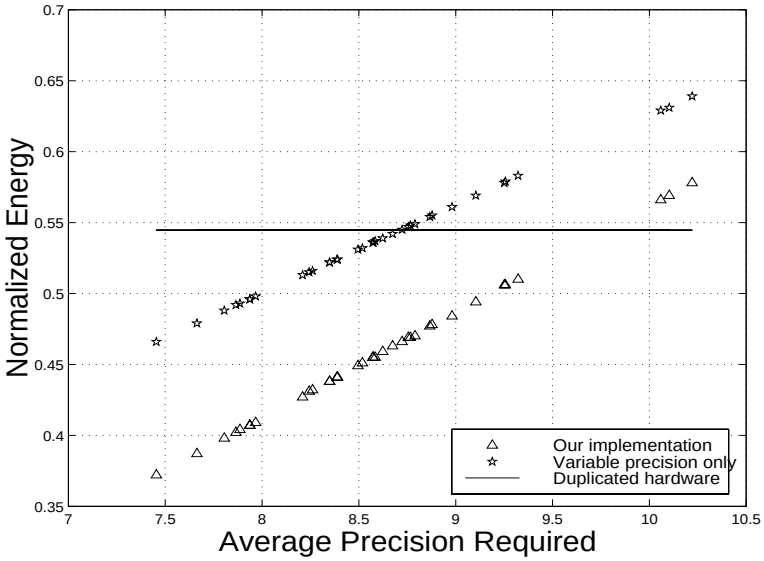


Figure 4-12: Average energy as a function of average precision

4.5 Conclusions

We have demonstrated an energy efficient DA based FIR filter architecture which uses two supply voltages and two RAC units to adapt itself to the immediate precision requirement of the data and perform just-in-time computation to meet a fixed throughput requirement. We have demonstrated a linear variation of average energy requirement with the average precision requirement. We have also demonstrated that 50% to 60% energy savings can easily be obtained in the case of speech data with little hardware overhead to the the fixed precision circuit.

Chapter 5
Conclusions

Energy efficient system design has become increasingly important. The presence of powerful and mature software development environments as well as the economics involved in having programmable solutions on general purpose processors rather than hardwired ones, have led to the proliferation of software solutions in the embedded systems domain. Pure hardware optimizations can no doubt decrease the power consumption significantly. However, it the software that finally runs on the hardware platform and therefore the overall system power consumption significantly depends on software. The choice of the algorithm can have an enormous impact on the energy consumption. Further, it is highly desirable that we structure our algorithms and systems in such a fashion that computational accuracy can be traded off with energy requirement. It is also important that we have robust energy models for software so that the system can predict the battery lifetime and make subsequent decisions about adjusting accuracy/throughput, processor voltage and frequency based on a required mission time.

In this thesis, the concept of energy aware software was introduced which incorporates an energy model with the corresponding application. Based on experiments conducted on the StrongARM SA-1100 microprocessor, a software energy macro-model was proposed which separates the leakage and switching energy components and shows less than 5% prediction error for a set of benchmark programs. Based on our experiments we conclude that as supply voltages and thresholds scale, leakage (static) components will become increasingly dominant. For the StrongARM, in continuous operation, the leakage energy is about 10% and increases exponentially with supply voltage and decreases linearly with operating frequency.

We have also introduced the notion of energy scalable computation in the context of signal processing. Algorithms that render incremental refinement of a certain quality metric such that the marginal returns from every additional unit of energy is diminishing are highly desirable in embedded applications. Using two broad classes of signal processing

algorithms we have demonstrated that using simple transformations (with insignificant overhead) the Energy-Quality (*E-Q*) behavior of the algorithm can be significantly improved. In general, we have concluded that doing the most significant computations first enables computational energy reduction without significant hit in output quality.

To obtain overall system energy scalability, all stages need to be scalable. For e.g. algorithmic transforms can ensure a certain energy-quality performance but if the datapath of the processor on which the algorithm runs can be scaled based on a *precision-on-demand* type approach, the system energy scalability can be dramatically improved. We have demonstrated an energy efficient DA based FIR filter architecture which uses two supply voltages to adapt itself to the immediate precision requirement of the data and perform *just-in-time computation* to meet a fixed throughput requirement. We have demonstrated a linear variation of average energy requirement with the average precision requirement. It can be concluded from our results that 50% to 60% energy savings can easily be obtained in the case of speech data with little hardware overhead to the fixed precision circuit.

It will be interesting to implement energy aware software on a variable voltage, variable supply, variable precision microprocessor with a smart operating system that allows dynamic voltage scheduling, just-in-time computation and efficient energy-accuracy trade-offs. We are implementing a scalable programmable sensor system on the StrongARM SA-1100 where the OS has such energy management features. From the hardware perspective, it will be very interesting to explore energy scalable datapaths such as variable bitwidth DSPs.

References

[1] J. Eager, “Advances in Rechargeable Batteries Spark Product Innovation”, Proceedings of the 1992 Silicon Valley Computer Conference, Santa Clara, Aug. 1992, pp. 243-253.

[2] C. Small, “Shrinking Devices puts the Squeeze on System Packaging”, EDN 39(4), Feb. 1994, pp. 41-46.

[3] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*, IEEE Press, 1998.

[4] T. Xanthopoulos and A. Chandrakasan, “A Low-Power DCT Core Using Adaptive Bit-width and Arithmetic Activity Exploiting Signal Correlations and Quantizations”, Symposium on VLSI Circuits, June 1999, pp. 11-12.

[5] J. Goodman, A. P. Dancy, A. P. Chandrakasan, "An Energy/Security Scalable Encryption Processor Using an Embedded Variable Voltage DC/DC Converter," IEEE Journal of Solid-state Circuits, vol. 33, no. 11, Nov. 1998, pp. 1799-1809.

[6] A. Sinha and A. P. Chandrakasan, “Energy Efficient Filtering Using Adaptive Precision and Variable Voltage”, 12th Annual IEEE ASIC Conference, Sept. 1999.

[7] D. Stepner, N. Rajan and D. Hui, “Embedded Application Design Using a Real-Time OS”, Proceedings of DAC 1999, New Orleans, pp. 151-156.

[8] 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.

[9] L. McMillan and L. A. Westover, “A Forward-Mapping Realization of the Inverse Discrete Cosine Transform”, Proceedings of the Data Compression Conference (DCC ‘92), March 1992, pp. 219-228.

[10] 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.

[11] A. Chandrakasan, et al., “Design Considerations for Distributed Microsensor Systems”, Proceedings of the IEEE 1999 Custom Integrated Circuits Conference, San Deigo, May 1999, pp. 279-286.

[12] 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.

[13] 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.

[14] 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.

[15] J. Sheu, et. al., “BSIM: Berkeley Short-Channel IGFET Model for MOS Transistors, IEEE Journal of Solid-State Circuits, SC-22, 1987.

[16] 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.

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

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

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

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

[21] S. Santhanam, et al., “A low-cost, 300-MHz, RISC CPU with attached media processor”, IEEE Journal of Solid-State Circuits, Nov. 1998, vol. 33, no. 11, pp. 1829-1839.

[22] J. Montanaro, et al., “A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor”, IEEE Journal of Solid State Circuits, Nov. 1996, vol. 31, no. 11, pp. 1703-1714.

[23] 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, pp. 707-713.

[24] A. Sinha and A. Chandrakasan, “Energy Aware Software”, Proceedings of the XIII Int’l Conference on VLSI Design, Calcutta, India, Jan 2000.

[25] J. T. Ludwig, S. H. Nawab and A. Chandrakasan, “Low-Power Digital Filtering Using Approximate Processing”, IEEE Journal of Solid-State Circuits, vol. 31, no. 3, March 1996, pp. 395-400.

[26] N. Ahmed, T. Natarajan and K. R. Rao, “Discrete Cosine Transform”, IEEE Trans. on Computers, vol. 23, Jan. 1974, pp. 90-93.

[27] W. H. Chen, C. H. Smith and S. C. Fralick, “A Fast Computational Algorithm for the Discrete Cosine Transform”, IEEE Trans. on Communication, vol. 25, Sept 1977, pp. 1004-1009.

[28] http://java.sun.com/products/jdk/1.2/docs/api/index.html.

[29] Yao, et. al., “Blind Beamforming on a Randomly Distributed Sensor Array System, IEEE Journal on Selected Areas in Communications, vol. 16, no. 8, Oct. 1998, pp. 1555-1567.

[30] A. Wang, W. Heinzelman and A. Chandrakasan, “Energy-Scalable Protocols for Battery-Operated Microsensor Networks”, Proceedings of SIPS ‘99, Oct. 1999.

[31] Stanley A. White, “Applications of Distributed Arithmetic to Digital Signal Processing : A Tutorial Review”, IEEE ASSP Magazine, July 1989, pp. 4-19.

[32] M. Mehendale, S. D. Sherlekar and G. Venkatesh, “Area-Delay Trade-off in Distributed Arithmetic based Implementation of FIR Filters”, X International Conference on VLSI Design, Jan. 1997, pp. 124-129.

[33] T. Xanthopoulos, “Low Power Data-Dependant Transform Video and Still Image Coding”, Ph.D. Thesis, Massachusetts Institute of Technology, Feb. 1999.

[34] R. Amirtharajah, T. Xanthopoulos and A. Chandrakasan, “Power Scalable Processing Using Distributed Arithmetic”, ISLPED99, Aug. 1999, pp. 170-175.

[35] A. P. Chandrakasan and R. W. Brodersen, “Minimizing Power Consumption in Digital CMOS Circuits”, Proceedings of the IEEE, vol. 83, no. 4, April 1995, pp. 498-523.

Appendix A

Energy Measurements

Taps	Time (s)	Freq (MHz)	V_{dd} (V)	I (A)	Charge (C)	Total Energy (Joules)	Energy/ Sample
1024	0.212	206	1.5	0.2215	0.046958	0.070437	6.88E-05
1024	0.225	192	1.5	0.2106	0.047385	0.0710775	6.94E-05
1024	0.245	177	1.5	0.1971	0.0482895	0.07243425	7.07E-05
1024	0.27	162	1.5	0.1835	0.049545	0.0743175	7.26E-05
1024	0.295	148	1.5	0.1693	0.0499435	0.07491525	7.32E-05
1024	0.33	133	1.5	0.1551	0.051183	0.0767745	7.50E-05
1024	0.37	118	1.5	0.1401	0.051837	0.0777555	7.59E-05
1024	0.42	103	1.5	0.1253	0.052626	0.078939	7.71E-05
1024	0.495	89	1.5	0.1101	0.0544995	0.08174925	7.98E-05
1024	0.595	74	1.5	0.0942	0.056049	0.0840735	8.21E-05
1024	0.74	59	1.5	0.0779	0.057646	0.086469	8.44E-05
1024	0.225	192	1.4	0.194	0.04365	0.06111	5.97E-05
1024	0.25	177	1.4	0.1817	0.045425	0.063595	6.21E-05
1024	0.27	162	1.4	0.1689	0.045603	0.0638442	6.23E-05
1024	0.3	148	1.4	0.1557	0.04671	0.065394	6.39E-05
1024	0.33	133	1.4	0.1423	0.046959	0.0657426	6.42E-05
1024	0.375	118	1.4	0.1287	0.0482625	0.0675675	6.60E-05
1024	0.425	103	1.4	0.1148	0.04879	0.06830	6.67E-05
1024	0.495	89	1.4	0.1006	0.049797	0.0697158	6.81E-05
1024	0.595	74	1.4	0.0861	0.0512295	0.0717213	7.00E-05
1024	0.74	59	1.4	0.0712	0.052688	0.0737632	7.20E-05
1024	0.225	192	1.35	0.1859	0.0418275	0.056467125	5.51E-05
1024	0.245	177	1.35	0.1739	0.0426055	0.057517425	5.62E-05

Table A.1: FFT energy measurements on the StrongARM SA-1100

Taps	Time (s)	Freq (MHz)	V_{dd} (V)	I (A)	Charge (C)	Total Energy (Joules)	Energy/ Sample
1024	0.27	162	1.35	0.1603	0.043281	0.05842935	5.71E-05
1024	0.295	148	1.35	0.1483	0.0437485	0.059060475	5.77E-05
1024	0.33	133	1.35	0.1361	0.044913	0.06063255	5.92E-05
1024	0.375	118	1.35	0.1231	0.0461625	0.062319375	6.09E-05
1024	0.42	103	1.35	0.1097	0.046074	0.0621999	6.07E-05
1024	0.495	89	1.35	0.0961	0.0475695	0.064218825	6.27E-05
1024	0.595	74	1.35	0.0821	0.0488495	0.065946825	6.44E-05
1024	0.74	59	1.35	0.0681	0.050394	0.0680319	6.64E-05
1024	0.25	177	1.3	0.1658	0.04145	0.053885	5.26E-05
1024	0.27	162	1.3	0.1541	0.041607	0.0540891	5.28E-05
1024	0.3	148	1.3	0.142	0.0426	0.05538	5.41E-05
1024	0.33	133	1.3	0.1301	0.042933	0.0558129	5.45E-05
1024	0.37	118	1.3	0.1175	0.043475	0.0565175	5.52E-05
1024	0.42	103	1.3	0.1047	0.043974	0.0571662	5.58E-05
1024	0.49	89	1.3	0.0916	0.044884	0.0583492	5.70E-05
1024	0.59	74	1.3	0.0781	0.046079	0.0599027	5.85E-05
1024	0.745	59	1.3	0.0647	0.0482015	0.06266195	6.12E-05
1024	0.265	162	1.2	0.1401	0.0371265	0.0445518	4.35E-05
1024	0.3	148	1.2	0.1294	0.03882	0.046584	4.55E-05
1024	0.33	133	1.2	0.1182	0.039006	0.0468072	4.57E-05
1024	0.37	118	1.2	0.1067	0.039479	0.0473748	4.63E-05
1024	0.425	103	1.2	0.0952	0.04046	0.048552	4.74E-05
1024	0.495	89	1.2	0.0831	0.0411345	0.0493614	4.82E-05
1024	0.595	74	1.2	0.071	0.042245	0.050694	4.95E-05
1024	0.74	59	1.2	0.0586	0.043364	0.0520368	5.08E-05
1024	0.33	133	1.1	0.1063	0.035079	0.0385869	3.77E-05
1024	0.37	118	1.1	0.0961	0.035557	0.0391127	3.82E-05
1024	0.425	103	1.1	0.0856	0.03638	0.040018	3.91E-05
1024	0.495	89	1.1	0.0748	0.037026	0.0407286	3.98E-05
1024	0.595	74	1.1	0.0638	0.037961	0.0417571	4.08E-05
1024	0.74	59	1.1	0.0526	0.038924	0.0428164	4.18E-05

Table A.1: FFT energy measurements on the StrongARM SA-1100

Taps	Time (s)	Freq (MHz)	V_{dd} (V)	I (A)	Charge (C)	Total Energy (Joules)	Energy/ Sample
1024	0.42	103	1	0.0765	0.03213	0.03213	3.14E-05
1024	0.49	89	1	0.0668	0.032732	0.032732	3.20E-05
1024	0.595	74	1	0.0569	0.0338555	0.0338555	3.31E-05
1024	0.74	59	1	0.0468	0.034632	0.034632	3.38E-05
1024	0.59	74	0.9	0.0503	0.029677	0.0267093	2.61E-05
1024	0.74	59	0.9	0.0414	0.030636	0.0275724	2.69E-05
64	0.008	206	1.5	0.2258	0.0018064	0.0027096	4.23E-05
128	0.0185	206	1.5	0.2266	0.0041921	0.00628815	4.91E-05
256	0.042	206	1.5	0.2267	0.0095214	0.0142821	5.58E-05
512	0.095	206	1.5	0.2265	0.0215175	0.03227625	6.30E-05
1024	0.212	206	1.5	0.2215	0.0480392	0.0720588	7.04E-05
2048	0.465	206	1.5	0.2218	0.105276	0.157914	7.71E-05
4096	1.015	206	1.5	0.2219	0.2298975	0.34484625	8.42E-05
1024	0.212	206	1.5	0.2215	0.046958	0.070437	6.88E-05
1024	0.225	192	1.35	0.1838	0.041355	0.05582925	5.45E-05
1024	0.25	177	1.27	0.1607	0.040175	0.05102225	4.98E-05
1024	0.27	162	1.2	0.1395	0.037665	0.045198	4.41E-05
1024	0.3	148	1.14	0.1209	0.03627	0.0413478	4.04E-05
1024	0.33	133	1.09	0.1045	0.034485	0.03758865	3.67E-05
1024	0.375	118	1.05	0.0905	0.0339375	0.035634375	3.48E-05
1024	0.425	103	0.97	0.0736	0.03128	0.0303415	2.96E-05
1024	0.495	89	0.92	0.0604	0.029898	0.02750616	2.69E-05
1024	0.59	74	0.85	0.0472	0.027848	0.0236708	2.31E-05
1024	0.74	59	0.8	0.0363	0.026862	0.0214896	2.10E-05

Table A.1: FFT energy measurements on the StrongARM SA-1100

Routine	Size	Time (s)	Freq (MHz)	Vdd (V)	I (A)	Energy/ Sample (J)
qsort	5000	0.87	206	1.5	0.1765	4.61E-05
	5000	1.02	177	1.3	0.1295	3.43E-05
dct	64	0.0004	206	1.5	0.2239	2.10E-06
	64	0.00046	177	1.3	0.1653	1.54E-06
idct	64	0.0005	206	1.5	0.2254	2.64E-06
	64	0.00059	177	1.3	0.1663	1.99E-06
fir	50000	0.0047	206	1.5	0.2331	3.29E-08
	50000	0.0055	177	1.3	0.1721	2.46E-08
log	1000	0.018	206	1.5	0.2258	6.10E-06
	1000	0.021	177	1.3	0.1663	4.54E-06
tdlms	32	0.083	206	1.42	0.2249	8.28E-04
	32	0.097	177	1.27	0.1728	6.65E-04

Table A.2: Energy consumption for some standard DSP routines on the SA-1100

Appendix B

Code

B.1 Assembly code for changing the StrongARM frequency

```
AREA Utility, CODE, READONLY
EXPORT setupARM

setupARM
   STMFD SP!, {r0-r1};
   MOV   r0, #0x17;
   SWI   0x123456;
   MRC   p15, 0, r1, c1, c0, 0;
   ORR   r1, r1, #0x1000      ; enables ICACHE
   ORR   r1, r1, #0x1         ; enables MMU
   ORR   r1, r1, #0x4         ; enables DCACHE
   ORR   r1, r1, #0x8         ; enables write buffer
   MCR   p15, 0, r1, c1, c0, 0;
   MOV   r0, #0x90000000      ;
   ADD   r0, r0, #0x20000     ;
   ADD   r0, r0, #0x14        ;
   LDR   r1, [r0]             ;
   BIC   r1, r1, #0xf         ;
   ORR   r1, r1, #0xa         ; Sets the frequency to 206.4 MHz
   STR   r1, [r0]             ;
   MRS   r0, CPSR             ; read the CPSR
   BIC   r0, r0, #0x1f        ;
   ORR   r0, r0, #0x10        ;
   MSR   CPSR, r0             ;
   LDMFD SP!, {r0-r1}         ;
   MOV   PC, LR               ;

END
```

B.2 Sample DSP code for energy measurement on the StrongARM

```
/* ---------------------------------------------------
   DSP routine power estimation on StrongARM SA-1100
   The Decimation-in-time FFT routine
   --------------------------------------------------- */

#include <stdio.h>
#include <math.h>
```

```
#include <stdlib.h>
#include <time.h>

#include “setuparm.h”    // To setup frequency and enable caches

#define ITER 1000        // Set the number of times the function is
                         // run to get stable current and run-time
#define PI 3.1415926536

void gen_random_data( float *data, int no );
void dit_fft( float *xr, float *xi, int n, float *Xr, float *Xi );
void print_array( float *data, int no );


main()
{
  const int nx=1024;      // Do a 1024 point FFT
  float xr[nx], xi[nx], Xr[nx], Xi[nx];
  int i;
  double diff_time;
  time_t start_time, end_time;

  setupARM();
  printf( “Setting up ARM .. done” );

  gen_random_data(xr,nx);    // Load with random data
  gen_random_data(xi,nx);

  time(&start_time);                // Main timer loop
  for( i=0; i<ITER; i++ ) {         // The function is run ITER times
    dit_fft( xr, xi, nx, Xr, Xi );  // and energy measurements
  }                                 // are made
  time(&end_time);

  diff_time = difftime(end_time,start_time);
  printf(“Time for %d iterations = %f\n\n”, ITER, diff_time );
  printf(“Time/ITERATION          = %f\n”, diff_time/ITER);


}


// The decimation in time FFT algorithm
void dit_fft( float *xr, float *xi, int n, float *Xr, float *Xi )
{
  int v, i, j, n1, index, rem, base, k;
  float wr, wi, tempr, tempi, a0, wr_new, wi_new;

  v = (int) (log(n)/log(2));
  a0 = 2*PI/n;

  for( i=0; i<n; i++ ) {
    n1 = n;
    index = 0;
```

```
    rem = i;
    for( j=0; j<v; j++ ) {
      n1 /= 2;
      if( (rem%2) == 1 ) index += n1;
      rem /= 2;
    }
    Xr[i] = xr[index];
    Xi[i] = xi[index];
  }

  index = 1;
  for( i=0; i<v; i++ ) {
    base = 2*index;
    n1 = n/base;
    for( k=0; k<n; k+=base ) {
      wr = 1;
      wi = 0;
      for( j=0; j<index; j++ ) {
   tempr = wr*Xr[j+k+index] - wi*Xi[j+k+index];
   tempi = wr*Xi[j+k+index] + wi*Xr[j+k+index];
   Xr[j+k+index] = Xr[j+k] - tempr;
   Xi[j+k+index] = Xi[j+k] - tempi;
   Xr[j+k] = Xr[j+k] + tempr;
   Xi[j+k] = Xi[j+k] + tempi;
   tempr = (float) cos(a0*n1);
   tempi = (float) -sin(a0*n1);
   wr_new = wr*tempr - wi*tempi;
   wi_new = wr*tempi + wi*tempr;
   wr = wr_new;
   wi = wi_new;
      }
    }
    index *= 2;
  }
}

void gen_random_data( float *data, int no )
{
  int i;
  for( i=0; i<no; i++ ) {
    data[i] = (float) rand();
  }

}
```