Low-Voltage Embedded Biomedical Processor Design

by

Joyce Y. S. Kwong

B.A.Sc. in Computer Engineering, University of Waterloo, 2004
S.M. in Electrical Engineering, Massachusetts Institute of Technology, 2006

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author .... [illegible]
Department of Electrical Engineering and Computer Science
May 21, 2010

Certified by .... [illegible] ....
Anantha P. Chandrakasan
Joseph F. and Nancy P. Keithley Professor of Electrical Engineering
Thesis Supervisor

Accepted by .... [illegible] ....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Low-Voltage Embedded Biomedical Processor Design

by

Joyce Y. S. Kwong

Submitted to the Department of Electrical Engineering and Computer Science on May 21, 2010, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

Advances in mobile electronics are fueling new possibilities in a variety of applications, one of which is ambulatory medical monitoring with body-worn or implanted sensors. Digital processors on such sensors serve to analyze signals in real-time and extract key features for transmission or storage. To support diverse and evolving applications, the processor should be flexible, and to extend sensor operating lifetime, the processor should be energy-efficient.

This thesis focuses on architectures and circuits for low power biomedical signal processing. A general-purpose processor is extended with custom hardware accelerators to reduce the cycle count and energy for common tasks, including FIR and median filtering as well as computing FFTs and mathematical functions. Improvements to classic architectures are proposed to reduce power and improve versatility: an FFT accelerator demonstrates a new control scheme to reduce datapath switching activity, and a modified CORDIC engine features increased input range and decreased quantization error over conventional designs. At the system level, the addition of accelerators increases leakage power and bus loading; strategies to mitigate these costs are analyzed in this thesis.

A key strategy for improving energy efficiency is to aggressively scale the power supply voltage according to application performance demands. However, increased sensitivity to variation at low voltages must be mitigated in logic and SRAM design. For logic circuits, a design flow and a hold time verification methodology addressing local variation are proposed and demonstrated in a 65nm microcontroller functioning at 0.3V. For SRAMs, a model for the weak-cell read current is presented for near-V_t supply voltages, and a self-timed scheme for reducing internal bus glitches is employed with low leakage overhead.

The above techniques are demonstrated in a 0.5-1.0V biomedical signal processing platform in $0.13\mu m$ CMOS. The use of accelerators for key signal processing enabled greater than 10× energy reduction in two complete EEG and EKG analysis applications, as compared to implementations on a conventional processor.

Thesis Supervisor: Anantha P. Chandrakasan
Title: Joseph F. and Nancy P. Keithley Professor of Electrical Engineering

Acknowledgments

Taking a drink from a fire hose is a humbling yet rewarding experience. Many people have helped me in this endeavour, but no one was more crucial than Prof. Anantha Chandrakasan. I am very grateful to Anantha for giving me a place in his group, for always challenging me, and for his constant encouragement and support. During my time here I have figured out the following: Anantha knows how to ask the key questions, and his judgement is always right, even if at first you disagree. What I have learned from him goes far beyond circuit design. Process technologies come and go, CMOS may become obsolete, but I will always remember his lessons on how to become a respected and successful professional.

I am equally grateful to my thesis committee members, Prof. George Verghese and Prof. Joel Dawson. They too, have asked the key questions and provided the insightful feedback that helped improve this thesis. I am grateful to Prof. Verghese for sharing his expertise on biomedical algorithms; his ideas have influenced the design of the biomedical processor in this thesis. I am grateful to Prof. Dawson for his advice not only on the technical aspects of this work but also on how to present the work in an effective way.

Texas Instruments has generously provided chip fabrication and funding that made this work possible. Further, I have had the great opportunity to collaborate with many people at TI. I am much indebted to Dr. Dennis Buss who always motivates us students with his enthusiasm and who is always generous with his support. Alice Wang was instrumental in finding the right person to answer our questions and getting our test-chips fabricated. But beyond that, she was a kind host each and every time I visited Dallas and made my stays much more enjoyable. Markus Koesler and Korbinian Huber helped me ramp up on MSP430 basics and provided invaluable feedback on my designs. Hans Moormann set up my summer internship in Freising which has been an unforgettable experience. During another summer internship at TI, I learned a great deal from discussions with Manish Goel and Michael DiRenzo and I look forward to working with them again.

I am indebted to several MIT colleagues for their help especially when I was starting to learn about biomedical applications. Thomas Heldt generously shared his time and insights with me back when I knew next to nothing about this field, and he deserves all the credit for pointing me to the WQRS algorithm which provided one of the key measurement results in this thesis. Eric Winokur taught me about analog front-ends and system design, and took care to keep me updated on his latest findings regarding MSP430 programming. Ali Shoeb and Eugene Shih were a great help in providing data, algorithms, and advice when I was porting the EEG algorithm to my chip. Last but not least, Prof. Sodini graciously agreed to help with my job search, and I am sure that having a job reference of his stature has helped me immensely!

One of the things I will miss the most about grad school is spending time with Ananthagroup members past and present. It has been a great privilege to be in a group where experts in so many different areas (including sports, Scotch tasting, and how to sell things on eBay) are a stone's throw away. But more importantly, everyone's collegiality and sense of humour make working in the lab highly enjoyable, perhaps despite the many long hours we spend there. Brian Ginsburg has bailed me out numerous times with his encyclopedic knowledge of UNIX, LaTeX and circuits, among other things. I would like to thank him for his sincere encouragement throughout my time in Ananthagroup. I have been extremely fortunate to have been able to work with Naveen Verma and Yogesh Ramadass on the 65nm chip and several papers. They are brilliant engineers who are generous with their time and brain power; I am especially grateful to Naveen for staying in the lab one late, late Friday night to help me get test results for ISSCC. I am grateful to Nathan Ickes for sharing his expertise with me on many occasions; I often wish I can do as many things as well as he does. Manish Bhardwaj has been a tremendous role model and I value his insightful and pragmatic advice. I thank Raul Blazquez for patiently answering all my questions on job search and beyond. I am grateful to Margaret Flaherty for her assistance with everything that makes the lab go 'round and for looking out for us grad students, informing us whenever there is free food or free stuff.

Ersin Sinangil has helped me in so many ways, starting from teaching me about SRAMs to generously offering me rides when we were in Dallas – I probably could not have survived without those trips to the grocery store! Masood Qazi has shared his insights with me in numerous areas, ranging from the serious (SRAM) to the practical (bartering on the reuse list) to the irreverent (poetry of the day). I have enjoyed discussions (formal and informal) with Marcus Yip, Patrick Mercier, and Jose Bohorquez, from whom I have learned a great deal. Lunch with Ananthagroup (Ben, Dave, Brian, Naveen, Denis, Viv, Pat, Marcus, Masood, Fred, Dani, Alex, Payam, among others) was an infinite source of laughter, gossip, silly ideas, and incredible tape-out stories. Thanks to all of Ananthagroup for livening up my day.

Vivienne Sze has been a great friend since the very beginning. It is only a small exaggeration to say that not a day goes by without us discussing, commiserating, or strategizing, and I credit her for motivating me to get up earlier and exercise more. I also need to say a big thank you to the SATC gals – Viv, Rumi, and Maryam – for their support, advice, the fun times at the pool / at brunch, and generally lending a sympathetic ear whenever I needed one.

I am grateful to my family for their patience and unwavering support, and for refraining from asking me when I will graduate. And finally, I believe my time at MIT would have turned out quite differently if I had not met Dennis. I thank him from the bottom of my heart for every difference he has made in my life, big and small, through his companionship, encouragement, advice, and help with math.

Contents

List of Figures

List of Tables

Acronyms

DSP Digital Signal Processor

ASIC Application-Specific Integrated Circuit

ADC Analog to Digital Converter

DMA Direct Memory Access

PMU Power Management Unit

FFT Fast Fourier Transform

CORDIC Coordinate Rotation Digital Computer

DCT Discrete Cosine Transform

ISR Interrupt Service Routine

FeRAM Ferroelectric RAM

ALU Arithmetic Logic Unit

SRAM Static Random Access Memory

VTC Voltage Transfer Characteristic

DIBL Drain-Induced Barrier Lowering

EKG Electrocardiogram

EEG Electroencephalography

FSM Finite State Machine

RISC Reduced Instruction Set Computer

Chapter 1

Introduction

Recent advances in sensor technology, low power circuits, and energy harvesting are providing growing opportunities for mobile electronic technologies to impact health care. One emerging application of great interest, conceptually shown in Figure 1-1, is ambulatory medical monitoring within a tele-health framework. In the envisioned scenario, body-worn or implanted sensor nodes acquire a subject's physiological signals and communicate the information to a local relay, such as a cell phone. The local relay then transmits this information through a wide-area network to a physician or a hospital, where the data can be further analyzed. This offers a major advantage in allowing a subject to be continuously monitored without being confined to the hospital and wired to various instruments. The subject is able to carry on with his/her daily activities, enabling physicians to assess the subject's condition in a more natural setting.

Ambulatory medical monitoring imposes stringent requirements on the sensor electronics and offers many opportunities for innovation in circuit technologies. Since sensor nodes for remote monitoring are worn over extended periods of time, comfort and convenience are important considerations. This implies that the nodes should be small and have long operating lifetimes, motivating the design of highly integrated and energy-efficient sensors and electronics. Moreover, because the monitoring takes place in uncontrolled environments, the quality of the data will be inferior to that collected in a hospital. Therefore, the sensor electronics should preserve sufficient signal fidelity in order for the physician to correctly interpret the data.

Arising from recent interest in this field, several sensor nodes with band-aid-sized form

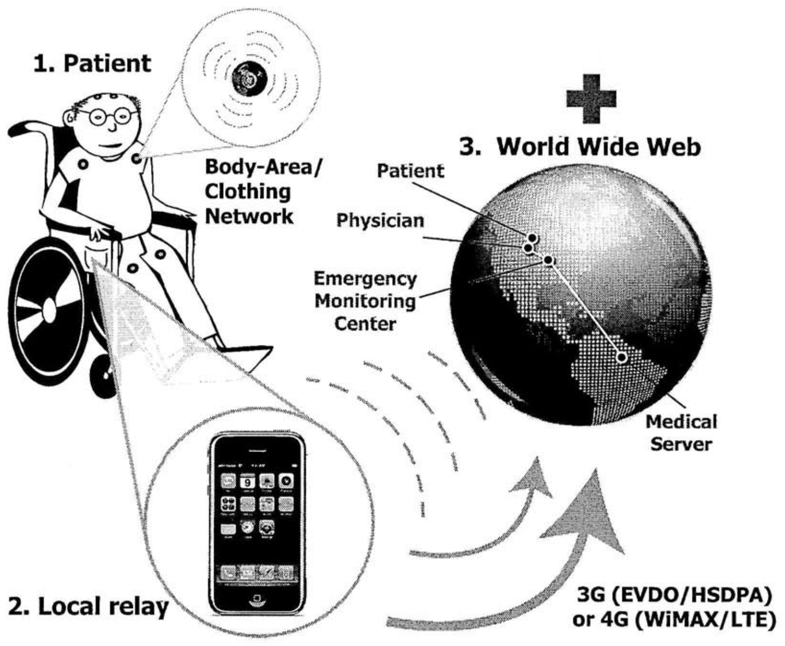

1. Patient
Body-Area/ Clothing Network
3. World Wide Web
Patient
Physician
Emergency Monitoring Center
Medical Server
2. Local relay
3G (EVDO/HSDPA) or 4G (WiMAX/LTE)

Figure 1-1: Ambulatory medical monitoring in a tele-health context, where body-worn or implanted sensors monitor a subject's vital signs and transmit observations through a local relay to a health care provider. (Figure courtesy of Patrick Mercier.)

factors have been demonstrated. For example, compact sensors are being developed by IMEC [1], Toumaz [2], Mindray [3], Corventis [4], and ST Microelectronics in partnership with Mayo Clinic. These systems focus on heart monitoring but some contain other sensors such as accelerometers or temperature sensors.

This thesis focuses on architectures and circuits for energy-efficient digital signal processing, with ambulatory medical monitoring as the target application. To provide a system-level context, we first describe a representative body-worn sensor node, of which the processor is a part. We then discuss related work on digital processing components on such sensor nodes.

1.1 Biomedical Sensor Nodes

A representative sensor node is shown in Figure 1-2. The sensor node contains an analog front-end to amplify and condition signals from one or several biomedical sensors, for example Electrocardiogram (EKG) electrodes, pulse-oximetry probes, and accelerometers. The analog front-end provides digitized signals to a local processor, which analyzes and prepares the data for storage or transmission. A communication module transmits the data over a short distance to a local relay such as a cell phone. Lastly, an energy subsystem contains the energy processing circuits to provide the power and voltage levels required by all components of the system. The system can be powered by several means – by a primary cell battery, rechargeable batteries and/or supercapacitors, or energy scavenging.

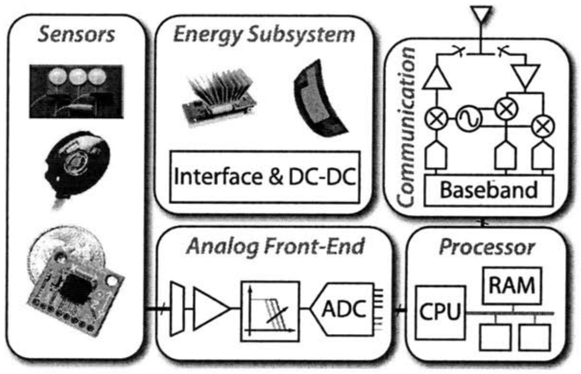


Figure 1-2: Block diagram of representative biomedical sensor node.

Table 1.1 lists the typical power consumption of the components of the sensor nodes, citing the state-of-the-art in research literature and commercial products. Substantial advances have been made in the analog domain; recent instrumentation amplifiers in the literature [5, 6] consume several microwatts of power, while low power Analog to Digital Converters (ADCs) consume less than a microwatt at 1-k samples per second [7, 8]. As seen from the cited examples, the power of digital signal processors varies greatly with the frequency and the amount of supported functionality. Likewise, power consumed for communication can vary widely, particularly between different modes of communication (e.g. wireless or wired). In interpreting the power of radios, it is important to note that they are often duty-cycled and turned on only during transmission or reception. Back-of-the-envelope calculations indicate that the duty cycle of the ChipCon radio [9] for an EKG application is on the order of 0.03%, assuming that only key information such as the heart rate is transmitted.

The data in Table 1.1 highlights how the digital signal processing component consumes a sizable proportion of the power in the sensor node electronics, especially after accounting for duty cycling of the radio. This raises the question of whether the sensor node should perform digital processing locally, or whether the raw physiological signals should

Table 1.1: Power consumption of state-of-the-art components in medical monitoring sensor node. Commercial products are indicated by *. [1]EEG stands for electroencephalography. [2]DSP stands for Digital Signal Processor.

Component	Reference	Active Power	Notes
Instrumentation Amplifier	Verma, VLSI 2009 [5]	$3.5\mu W$	for EEG[1] acquisition
	Denison, ISSCC 2008 [6]	$6\mu W$	for neuroprosthesis
	Yazicioglu, JSSC 2008 [10]	$6.9\mu W$	for EEG acquisition
	Bohorquez, VLSI 2010 [11]	$\approx 1.3\mu W$	reconfigurable for different bio-signals
	INA333* [12]	$90\mu W$	Micropower Instrumentation Amplifier
ADC	Verma, ISSCC 2006 [7]	$25\mu W$	12-bit, 100kS/s
	Agnes, ISSCC 2008 [8]	$3.8\mu W$	12-bit, 100kS/s
	ADS7866* [13]	$220\mu W$	12-bit, 100kS/s
DSP[2]	Jocke, VLSI 2009 [14]	$0.72\mu W$	8-bit microcontroller, 475kHz
	Phyu, A-SSCC 2009 [15]	$176\mu W$	Custom EKG QRS (heart rate) detection chip, 1MHz
	MSP430F5438* [16]	$363\mu W$	16-bit microcontroller, 1MHz
Radio	Mercier, JSSC 2009 [17]	4.36mW	3-5GHz Pulsed UWB transmitter, 15.6MBaud Tx, -16.4dBm output power
	ChipCon 2550* [9]	33.6mW	2.4GHz transmitter plus packet handling logic, 500kBaud Tx, -12dBm output power

be transmitted elsewhere for processing. One reason to include local processing is to allow basic monitoring directly on the subject's body, so that the subject does not need to carry a local relay at all times or be within the range of a basestation. Further, in systems like loop recorders [18] which store data in memory for later download, local processing is necessary to identify irregular episodes for recording and thereby reduce memory storage requirements. For a quantitative illustration, we analyze the power corresponding to two scenarios with and without local processing, using an epilepsy detection application described in [19] as an example. The first sensor node listed in Table 1.2 transmits the raw captured Electroencephalography (EEG) channel to a basestation for analysis. The second node processes the EEG signal and extracts several features (consisting of 7 words) every two seconds and transmits only the features to the basestation for classification. It is seen that local processing can sufficiently reduce the amount of data transmitted to lower the overall system power. Of course, power in the Digital Signal Processor (DSP) must be kept low relative to the radio power in order to yield net power savings. Accordingly, this thesis focuses on the design of a low power DSP for ambulatory monitoring applications, both to realize power reductions like in the previous example, and more generally to reduce the proportion of power consumed by the DSP component of a sensor node.

Table 1.2: Computation versus communication power trade-off in sensor nodes, showing the benefit of local processing to reduce radio transmit power.

Component	No local processing	Local processing
Amplifier	$3.5\mu W$ [5]	$3.5\mu W$
ADC	$3.8\mu W$ [8]	$3.8\mu W$
DSP	0	$19\mu W$ (Chapter 6)
Radio: Transmit	4kbps×40nJ/bit[5]	56bps×40nJ/bit
Start-up	$4.8\mu W$	$4.8\mu W$ every 2 sec
Idle Mode	$0.46\mu W$	$0.46\mu W$
Total	$176.4\mu W$	$31.4\mu W$

Biomedical applications are particularly amenable to low power processing due to the relatively low bandwidths of physiological signals. Table 1.3 lists the sample rate and resolution of several sources of physiological data. The sample rates are typically less than 1kHz, leading to relaxed speed requirements for analog-to-digital conversion and digital processing. It should be noted, however, that certain types of signals such as the EEG have low amplitudes on the order of microvolts, which impose stringent noise requirements on the analog front-end amplifier.

1.2 Related Work in DSPs for Medical Monitoring

As a result of recent interest in ambulatory medical monitoring, researchers have demonstrated numerous prototype monitoring systems. This section reviews relevant work in the DSP component of such systems to provide context on the state-of-the-art. The digital processors found in these systems span a spectrum ranging from commercial high-performance DSPs, general-purpose low power processors, to Application-Specific Integrated Circuits (ASICs) customized for a specific function. At one end of the spectrum, many prototype systems in the literature employ off-the-shelf processors. For instance, a 32-bit DSP was used in [22] to build a mobile acquisition system for monitoring motor

Table 1.3: Typical sample rates and resolutions of of physiological signals.

Signal	Source	Sample Rate	Resolution	Signal Amplitude
EKG	MIT-BIH Arrhythmia Database	360Hz	11-bits	1-5mV
EKG	Long-Term ST Database	250Hz	12-bits	1-5mV
EKG	ANSI/AAMI EC13 Test Waveforms	720Hz	12-bits	1-5mV
EEG	Data for seizure onset detector development [19]	256Hz	16-bits	1-100μV
Photoplethysmograph	MIMIC Database [20]	125Hz	12-bits	1mV
Gyroscope	Tremor Monitoring [21]	200Hz	12-bits	several volts

control. Similarly, a 32-bit floating point DSP operating at 150MHz was utilized in [23] for real-time epileptic seizure onset detection from a set of EEG signals. While these powerful processors are convenient for system prototyping, it is likely that many applications do not actually require their high performance or advanced capabilities, and can instead achieve lower power through slower processors with appropriate hardware support for specific computations. For example, part of the algorithm in [23] was subsequently implemented on fixed point custom hardware operating at 75Hz [5].

Several systems employ commercial low-power microcontrollers such as the 8051 [2] and MSP430 [1]. Further, the design of ultra-low power processors has been an active area of research. These processors are not specifically targeted towards biomedical applications but can nonetheless be used in a sensor node. Some such as [14, 24, 25, 26] are designed to achieve very low energy per instruction through aggressive voltage scaling, while [27] was optimized for low leakage power. Generally speaking, these processors achieve moderate performance (tens of MHz or less) and support general-purpose instruction sets. Although compact, this type of processor lacks dedicated hardware such as multipliers and filters to support efficient signal processing. To address this deficiency, [28] presented a micropower DSP with a 16-bit CPU along with FIR filter and Fast Fourier Transform (FFT) modules to help reduce energy in microsensor applications. Similarly, the processor in [29] featured a 32-bit ARM CPU and an FFT module.

On the other end of the spectrum lie custom ASICs that are hardwired for specific functions. For instance, the EKG signal processor in [30] featured a custom circuit to compress EKG signals for storage. In [15], the authors proposed extracting the peaks of an EKG signal from its wavelet transform, and designed hardware to compute the transform and perform adaptive thresholding. Similarly, the EEG acquisition system-on-chip in [5] contained a hardwired modulated filter bank to extract energy from different frequency bands in the signal.

Figure 1-3 plots the energy per operation of the previous work cited above. For DSPs and low-power processors, an operation refers to an instruction. For the ASIC of [5] which implements a series of FIR filters, an operation refers to one multiply-accumulate in an FIR filter. It is seen that floating-point DSPs and even commercial microcontrollers consume several hundred pJ/op. Custom, ultra-low-power microcontrollers in the literature achieve tens to several pJ/op. Finally, since ASICs only include circuits for the specific computation at hand, they typically consume less energy relative to equivalent implementations on a general-purpose processor.

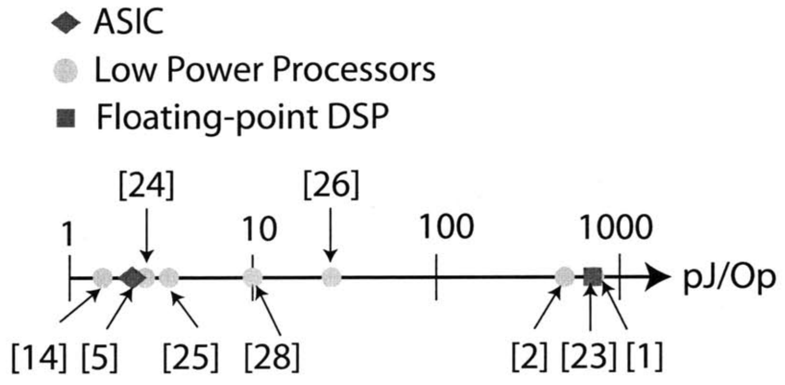


Figure 1-3: Energy per operation (an instruction or a signal processing operation) in related work.

1.3 Signal Processing Platform Overview

As seen in Section 1.2, previous work has focused on system prototypes with off-the-shelf DSPs, general-purpose low power processors, or custom ASICs implementing a specific algorithm. Development of a flexible but low power platform for biomedical signal processing has received relatively little attention. Motivated by the diverse applications in this field and active efforts in algorithm development, this work proposes a programmable platform targeted for ambulatory medical monitoring. We first give an overview of the processor and its key features. In the subsequent chapters we will analyze different aspects of the processor, ranging from system-level architecture to transistor-level circuit design.

In the spectrum of processor designs discussed above, our design lies between a low power CPU and a custom ASIC. The processor, whose block diagram is shown in Figure 1-4, features a 16-bit CPU based on Texas Instruments' MSP430 for general-purpose computation and control. Importantly, it includes four accelerators customized to efficiently perform common signal processing operations. As a platform, this processor must support applications of varying complexity and performance demands in an energy-efficient manner. The system is therefore voltage-scalable from 1V down to 0.5V, such that low performance applications can be executed at 0.5V to reduce energy. Further, each module can be dynamically clock- and power-gated when not in use. The clock control block at the top level distributes a clock to a module upon request, and disables the clock after the module has completed processing. In addition, 15 power domains are implemented with on-chip, high-V_t switches controlled by the power management unit.

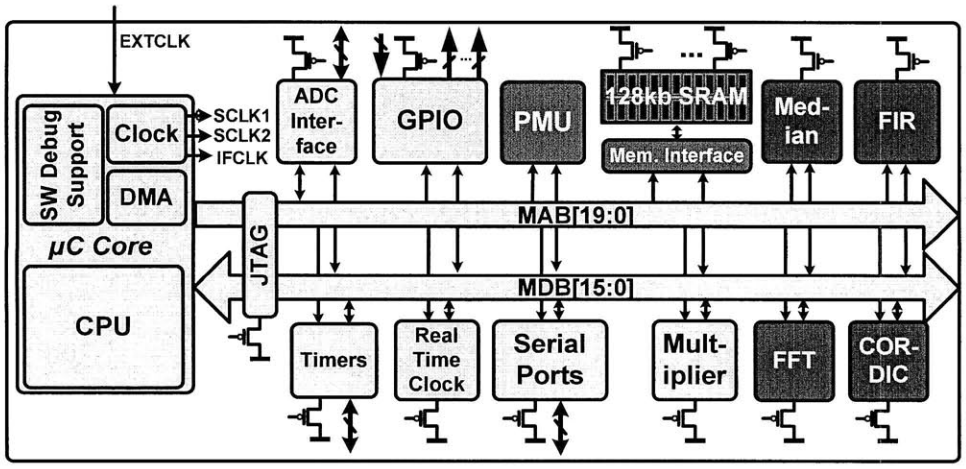


Figure 1-4: Block diagram of signal processing platform. Darkly shaded blocks are custom-designed for biomedical applications and low-voltage operation.

The lightly shaded blocks in Figure 1-4 are based on components from Texas Instrument's MSP430 microcontroller, a low power microcontroller widely used in industry. We custom-designed the darkly shaded blocks for biomedical applications and low-voltage operation. A description of the modules on this platform is provided in Chapter 6.

ducing energy and propose solutions. Within those areas, this thesis makes the following contributions:

1. *Accelerator architectures.* There is a wealth of literature on algorithms for ambulatory medical monitoring utilizing diverse signal processing techniques. It is well-known that hardware implementations of such techniques can be made more energy-efficient than software realizations. However, hardware accelerators must be broadly applicable across many usage scenarios, while at the same time satisfying stringent area and power constraints. Chapter 2 presents four hardware accelerators – a median filter, a CORDIC engine, an FFT module, and an FIR filter – chosen for their impact on many applications. Although there has been much previous work on these architectures, we find ways to extend the classic structures to support more usage scenarios at low hardware cost. We propose improvements to the convergence range and quantization error in the CORDIC engine, low power optimizations in the FFT and FIR filter, as well as cycle count optimizations in the latter. While motivated by biomedical applications, these improvements are applicable to signal processing architectures in general. The proposed accelerators are demonstrated in a prototype test-chip with which we can quantify the cycle count and energy reduction afforded by the accelerators. Most importantly, we show that accelerators enable greater than 10× energy reduction in several complete applications compared to a conventional processor.

2. *Low-voltage digital circuit design.* A key strategy for improving the energy efficiency of the processor is to lower its supply voltage until application performance constraints are barely met. However, in designing low-voltage logic and SRAM circuits, we must contend with their increased sensitivity to manufacturing process variation. For logic design, Chapter 4 proposes an approach to verify timing constraints while accounting for significant variation at low voltages. The approach was demonstrated in a 65nm MSP430 test-chip functioning down to 0.3V, which was the first processor to achieve deep sub-threshold operation at the 65nm node.

   For SRAM design, Chapter 5 proposes a model for the worst case read-current, a key parameter in SRAM design, when the supply voltage is near the transistor threshold and conventional models do not apply. We analyze the energy trade-offs of two techniques that enable writing in the face of variation. The analysis reveals that significant energy is expended in driving a large data bus connecting sub-blocks within an SRAM. Accordingly, we employ a self-timed scheme to reduce glitches on the data bus. While the usual practice in SRAM design is to anticipate the worst case and eliminate all glitches, we find this imposes excessive leakage overhead. Instead, we implement a scheme that considers the average case and removes most glitches at a small fraction of the leakage cost.

3. *System architecture for low power processors.* The addition of accelerators in particular, and functional modules in general, imposes costs which must be considered when designing the top-level system architecture of the processor. Specifically, accelerators increase the capacitive load on the system bus. Accordingly, in Chapter 3 we analyze an alternate dual-bus structure and investigate the optimal assignment of modules to buses to minimize bus energy. In addition, accelerators introduce leakage power which must be reduced through power gating. Chapter 3 concludes with a discussion of power gating implementation and trade-offs.

Chapter 2

Accelerator Architectures

Accelerators, or hardware dedicated to a specific function, have been designed into many electronic systems ranging from the Intel 8087 floating point co-processor first announced in 1980 to video encoder/decoders integrated into modern multimedia processors for mobile handsets. In the context of sensor processors, it has been shown in [28] that accelerators can provide considerable energy savings. This chapter first discusses the selection of hardware accelerators based on algorithm requirements in the ambulatory monitoring domain, then describes the accelerator architectures and proposed improvements.

2.1 Accelerators for Biomedical Signal Processing

A wide range of techniques has been applied to the processing of biomedical signals. To highlight some representative examples, Table 2.1 lists the main signal processing operations in a number of published applications for ambulatory medical monitoring. This guided the selection of common signal processing tasks that can benefit from hardware acceleration.

From this list, it is clear that several operations are often found in ambulatory monitoring applications. Filtering is a prevalent task since signal acquisition typically introduces noise which must be removed. Wavelet decompositions have been employed in several of the listed application domains. The FFT is needed in several instances to analyze the frequency content of various physiological signals. We further observe that many applications involve comparing a signal against a threshold, for which we require the n^{th} largest or smallest sample in a window of data. Similarly, median filtering, which is helpful for removing noise spikes without degrading signal edges (unlike a low pass filter), requires the median of a

Table 2.1: Signal processing in ambulatory medical monitoring applications.

Signal	Application	Operations
EKG	QRS Detection [31]	IIR filter (M=16, N=32) FIR filter (N=128) Adaptive threshold detection
EKG	QRS Detection [32]	IIR filter (M=2, N=10) Curve length transform $\sum_{k=i-w}^{i} \sqrt{\Delta t^2 + \Delta y_k^2}$
EKG	QRS Detection [33]	IIR filter (M=2, N=12) FIR filter (N=4) x^2 Median filter (N=10)
EKG	Analysis of R-R interval [34]	FFT
EKG	QRS detection [15]	Wavelet transform Threshold comparison
EKG	Arrhythmia classification [35]	Integer division Sum of absolute differences
EEG	Epileptic seizure onset detection [23, 36]	Wavelet decomposition
EEG	Indicators of sleepiness during driving [37]	$H(p) = -\sum_k p_k \log p_k$
EEG	Elimination of Periodic ECG Artifacts [38]	$\psi(x[n]) = x^2[n] - x[n+1]x[n-1]$ FIR filter
Photoplethysmograph	Finding blood oxygen saturation [39]	FIR filter (N=22) $\log x$
Heart Sound	Auscultation aid [40, 41]	FFT atan, sin, cos inverse FFT

window of data. This suggests that a means for sorting a block of data samples would be useful. We also note that algorithms utilize a range of basic mathematical functions such as division, $\log x$ and $\sqrt{x}$ that are not natively supported in fixed point microcontrollers, and thus require expensive software emulation.

To understand what percentage of clock cycles in typical applications are spent on these signal processing operations versus other tasks, several applications were profiled on a standard MSP430 microcontroller with a 16-bit RISC CPU and hardware multiplier. Open source code in C or MATLAB was available for the chosen applications and thus simplified our task. First, the applications were ported to the MSP430 and compiled using a standard MSP430 C compiler, which unfortunately did not have profiling capabilities. Instead, we executed the resulting instructions on an MSP430 CPU in cycle-accurate VHDL simulations and recorded the number of clock cycles spent in major portions of the applications.

The cycle breakdown of algorithms for EKG arrhythmia classification [35], pulse-oximetry [39], and heart sound processing [40, 41] are shown in Figure 2-1. It should be noted that the code was not hand-optimized for the MSP430 (for example, the heart sound application was in floating point), but default compiler optimizations were enabled. The arrhythmia classification algorithm involves many control tasks that are suited to a general-purpose CPU (e.g. loops, comparisons, searches). However, many clock cycles are consumed while matching an incoming heart beat against templates. Part of this pattern matching involves computing the sum of absolute differences (SAD). It is possible to replace SAD with matched filters, which can then be mapped to a hardware FIR module. FIR filtering contributes the majority of clock cycles in the pulse-oximetry application as well. In the third algorithm, heart sounds are transformed into the frequency domain, processed, then transformed back into the time domain. Consequently, the FFT/IFFT is a key component consuming roughly half of the total number of clock cycles.

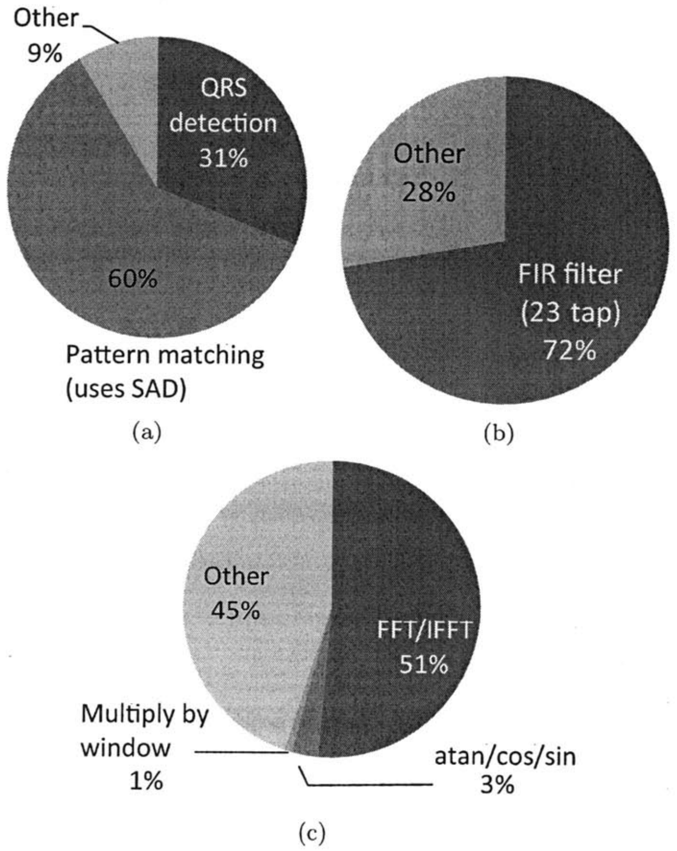


Figure 2-1: Breakdown of cycle count on a standard MSP430 while it executes (a) EKG arrhythmia classification, (b) pulse-oximetry, and (c) heart sound processing.

The profiling results illustrate that on a general-purpose CPU, operations such as filtering and FFT can dominate the cycle count in an application. To determine the potential gains from implementing these operations in hardware, we estimate the associated hardware complexity and the achievable cycle count savings, since both metrics can be computed before design time. In Table 2.2, as a measure of hardware complexity, we list the primary arithmetic blocks required in a given accelerator and architecture. The cycle count savings are obtained by profiling operations on the 16b general-purpose CPU to obtain a baseline cycle count, then calculating the theoretical number of cycles required on a given hardware accelerator architecture.

Table 2.2: Evaluation of accelerator costs and benefits before platform design.

Operation	Cycle Count (CPU + HW Multiplier)	Cycle Count (Accelerator)	Arithmetic Blocks in Accelerator
N-tap FIR	53N + 182	N	1 mult., 1 adder
M-level wavelet decomposition	$(2^M - 1)$(53N + 182)	2^M	$2MN$ mult., $2M(N-1)$ adders
512-point FFT	918880	6431	1 complex mult., 2 complex adders, 1024-word memory
65-point median filter	1210	6	130 adders, 65 registers
$\ln(x)$	4214	50	3 adders, 3 barrel shifters

From Table 2.2, it is clear that accelerators significantly reduce the number of clock cycles for the above operations. From an energy perspective, the total energy to complete an operation can be expressed as energy/cycle $\times$ number of clock cycles. Therefore, an accelerator can reduce computational energy as long as its energy/cycle is not much larger than that of a CPU. Based on the hardware complexity listed in Table 2.2, this is a reasonable assumption for most of the accelerators except the wavelet filter bank (a series of tapped delay lines with one multiplier per tap). However, instead of adding hardware to implement all filters in the wavelet filter bank, the hardware cost can be reduced by using one filter iteratively, if the additional latency is acceptable.

Based on the above observations, we have decided to include four accelerators in the signal processing platform: a median filter, a Coordinate Rotation Digital Computer (CORDIC) engine, an FFT module, and an FIR filter. Understandably, a large body of work exists on hardware architectures for filtering and FFT due to their importance in signal processing. While circuit realizations of other functions are less well-studied, several implementations have also been demonstrated in prior work. In this chapter we focus on augmenting classic architectures to support the wide range of use cases found in monitoring applications. Further, we propose several optimizations to reduce power and improve speed. These optimizations are not specific to biomedical algorithms and are also applicable to other fields. For example, FIR filtering and FFT are found in monitoring applications for sensor networks [42]. By virtue of the accelerators, the biomedical processor itself can also be leveraged for other non-biomedical applications with a sizeable signal processing component.

2.2 Median Filter

As mentioned previously, a way to sort a block of data is useful for several reasons in biomedical applications. The minimum and maximum are often used in setting adaptive thresholds, while the median is needed in median filtering (the output of such a filter is the median value in a window of data samples). Median filters are commonly used in image processing to remove speckle noise, with the advantage that it preserves edges in the image. Similarly, median filtering is employed in biomedical signal processing to remove noise spikes without degrading edges in the signal. It is also useful for removing baseline wander from biomedical recordings.

Since a hardware median filter essentially contains circuitry to sort a window of data, the area of the filter can be quite large for long windows. An architecture with a relatively small area was selected for the biomedical processor. This section first describes the architecture, then focuses on analyzing the trade-off between area, energy, and leakage of the median filter. An optimization to shorten the filter's critical path is also presented.

2.2.1 Median Filter Design

Several architectures have been proposed for median filtering in hardware, and a comprehensive overview can be found in [43]. Among these, the sorted list architecture described in [44, 45] is attractive for a low power processor because its hardware complexity grows linearly with the number of samples in a window of data. This is in contrast to other architectures such as [46, 47, 48, 49], whose complexity grows quadratically with the window length.

Illustrated in Figure 2-2, the sorted list architecture maintains a window of data in a shift register and in sorted order. Each data sample is also associated with a counter that maintains the "age" of the sample, or how many clock cycles it has been stored in the window. When an incoming sample arrives, the oldest sample is removed from the window. The incoming sample is then compared against all samples in the window before being inserted in the correct position. When all the comparisons are done in parallel, the computation takes a constant number of clock cycles regardless of window length.

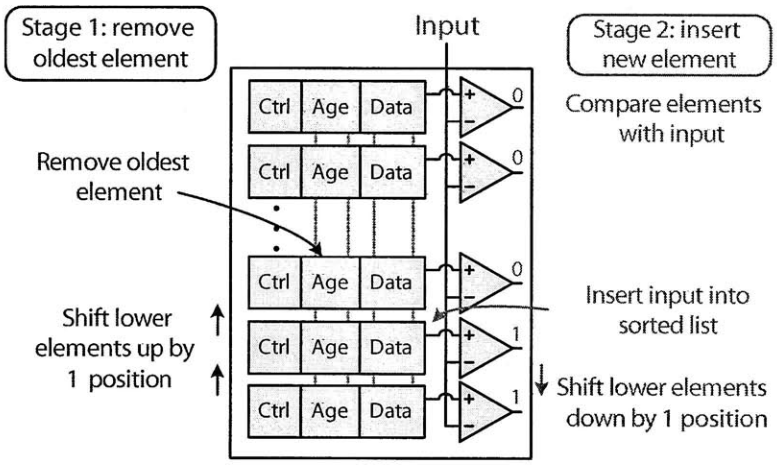

Stage 1: remove oldest element
Input
Stage 2: insert new element
Ctrl
Age
Data
Compare elements with input
Remove oldest element
Shift lower elements up by 1 position
Insert input into sorted list
Shift lower elements down by 1 position
0
0
0
1
1

Figure 2-2: Median filter based on sorted list architecture.

As seen from Figure 2-2, a basic processing element (PE) is replicated N times for a median filter of length N. The PE is thus the key determinant of the area, leakage, and performance of the filter. Details of the PE are shown in Figure 2-3. The PE contains a register (*data*) to store the value of a data sample. For a median filter of length N, the PE requires a register with $log_2(N)$ bits to store the index (*idx*), which indicates how long the data sample has resided in the filter.

When a new input sample arrives, the *idx* register is incremented. In the PE where $idx == N$, the corresponding data sample is removed by shifting the contents of all the PEs below upwards by one position. This is indicated by asserting the *shiftUp[i]* signal, which propagates through an OR chain to the downstream PEs. The *shiftUp* causes the PEs to move *idxBelow* and *dataBelow* into the *idx* and *data* registers.

After removing the oldest data sample, the next step involves inserting the new input

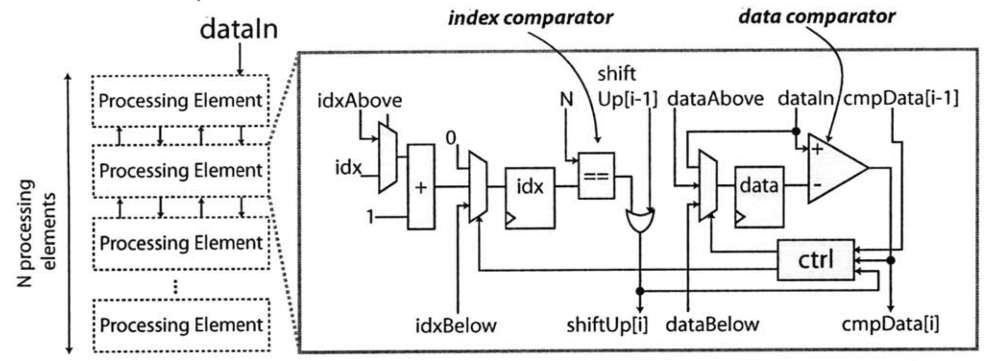


Figure 2-3: Reference architecture for median filter and details of the processing element.

sample *dataIn* into the sorted list. In the basic design, *dataIn* is compared with the *data* register in every PE. *dataIn* is inserted into the i^{th} PE in which $dataIn >= data[i]$ and $dataIn < data[i-1]$, and these two conditions are encoded in *cmpData[i] and cmpData[i-1]* respectively. Contents of the downstream PEs are all shifted down by one position by having each PE store *idxAbove* and *dataAbove* into its registers.

This architecture can find the median (and any other order statistic) in very few clock cycles by virtue of its parallel structure. Correspondingly, it occupies a large area and consumes significant leakage power when not power-gated. To reduce area in our median accelerator, we share the data comparator amongst several processing elements in a time multiplexing scheme. Figure 2-4(a) shows the synthesized area of the median filter when a data comparator is shared amongst 2, 4, 8, and 16 PEs. The area savings reach diminishing returns since other components in the PE start to dominate the area.

We further optimize the processing element by shortening the critical path of the design. The first way to shorten the critical path is by performing the index and data comparisons on separate clock cycles. Otherwise, if they are performed on the same clock cycle, the critical path would pass through all PEs.[1]

In another modification to the reference architecture, we share one index comparator between several PEs in the same way that the data comparator is shared. As illustrated in Figure 2-4(b), this reduces the area by approximately 20%.

With the above modifications, the critical path in the design lies in the long chain of OR gates that generates the *shiftUp[i]* signals. A chain of 2-input OR is necessary since each

[1]This is because we must wait until the index comparison completes and the *shiftUp* signal propagates through all PEs, before we can compare the appropriate data to the incoming sample.

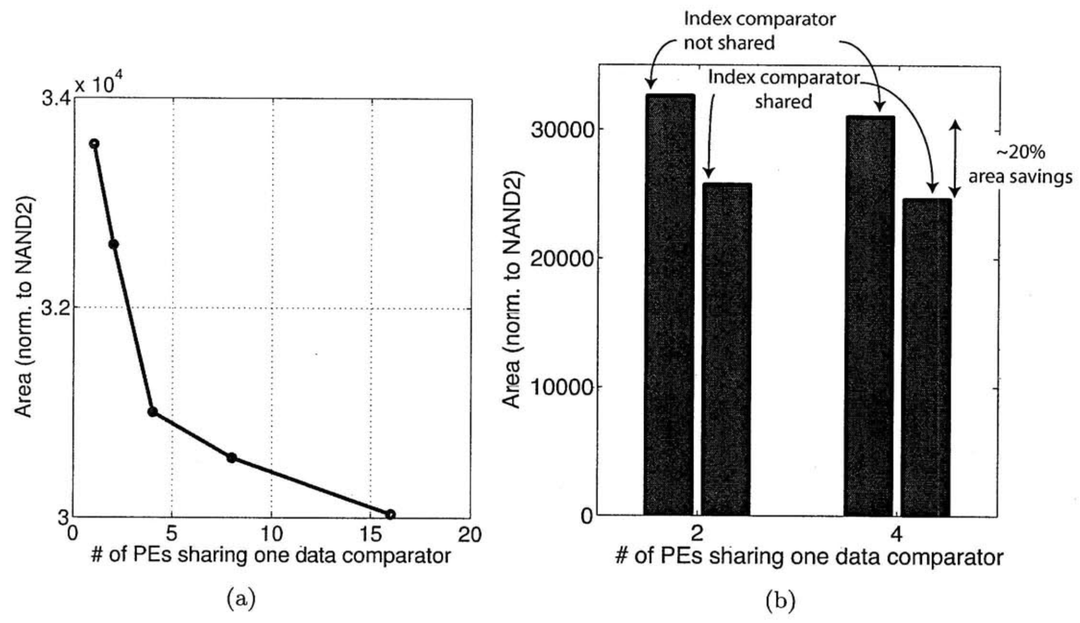


Figure 2-4: (a) Area savings from sharing data comparator between several PEs. (b) Additional area savings from sharing index comparators between several PEs.

gate's output is used by a processing element to generate local control signals. However, the effective critical path length can be shortened using a bypass technique similar to that in carry bypass adders [50]. This is illustrated in Figure 2-5. For every group of 4 processing elements, a group *shiftUp* signal is generated with a 5-input OR gate. This shortens the critical path from 64 2-input ORs to 15 5-input ORs plus 4 2-input ORs.

In the final median filter design, the degree of comparator sharing is taken at the knee of the curve in Figure 2-4(a) to balance the opposing trends of reducing area and increasing latency. In this implementation, one index and one data comparator are time-multiplexed between four PEs, implying that one median computation requires $4 \times 2 = 8$ clock cycles. Nevertheless, this is substantially faster than equivalent software implementations that involve either sorting a block of data or maintaining a sorted linked list.

2.2.2 Area, Energy, and Latency Trade-offs

The decision to share one comparator among several PEs embodies a trade-off between the area, energy, and latency of the median filter. The following lists the differences between two designs in which a) several PEs share a comparator, and b) each PE has its own comparators. We refer to these as Design A and B respectively.

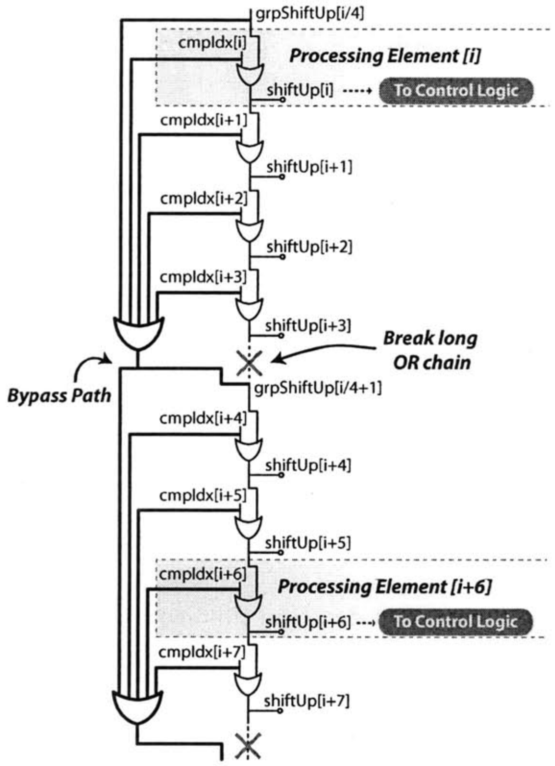


Figure 2-5: Bypass technique to shorten critical path in median filter. The bypass path is indicated by thicker black lines.

- *Latency:* If x PEs share one comparator in Design A, a median filtering operation takes $2 * x$ cycles to complete.
- *Switched capacitance:* Design A has fewer logic gates and lower interconnect capacitance. The net effect is that Design A has lower effective switched capacitance as will be shown by simulation results below.
- *Area:* Design A has smaller area.
- *Leakage power:* There are two opposing factors to consider. Design A has fewer standard cells. However, Design B can function at a lower V_{DD} while achieving the same latency as Design A. These two effects will be evaluated in the discussion below.

This section investigates the above trade-offs by simulating median filters with different degrees of comparator sharing. This is done via gate-level simulations of the designs after layout, with accurate switching activity and extracted wiring parasitics. Since a median filter would typically be part of a larger system, we examine the trade-offs in the context of two different system scenarios illustrated in Figure 2-6. In the first case, the median filter does not have an independent power supply. This implies that the supply voltage (V_{DD}) is at a fixed level and the filter cannot be powered gated during its idle periods. In the second case, the median filter has its own power supply. This implies that the filter can be power-gated during idle periods, and its operating voltage can be set independently from the rest of the system.

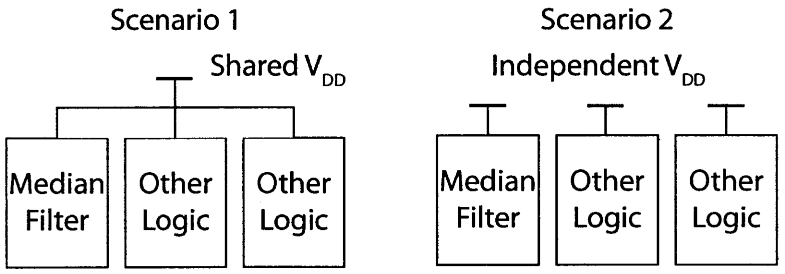


Figure 2-6: Two operation scenarios analyzed in this section. In scenario 1, the median filter shares a power supply with other logic. In scenario 2, the median filter has an independent power supply.

Scenario 1: No independent power supply

Figure 2-7 illustrates the power consumption over time of the median filter in scenario 1. The variables are defined as follows:

- P_{act}: the active power when the median filter is computing the median. This equals the total simulated power minus the leakage power.
- T_{cyc}: the clock period (100ns in this case).
- N_{cyc}: the number of clock cycles needed to find the median. In a design with no comparator sharing, N_{cyc} equals 2. In a design where x PEs share one comparator, $N_{cyc} = 2x$.
- T_{acc}: the time between the arrival of consecutive input samples.
- P_{leak}: the leakage power of the median filter when it is not active (and not power gated since it shares a supply with other logic).

The total energy per output sample, which varies with T_{acc}, is given by:

$$E_{total} = P_{act}T_{cyc}N_{cyc} + P_{leak}T_{acc} \tag{2.1}$$

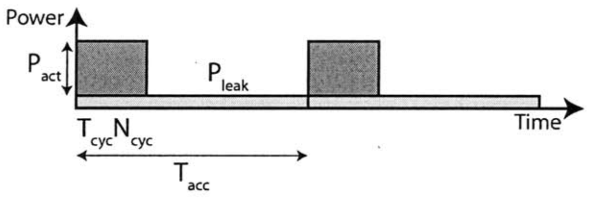


Figure 2-7: Power profile without power gating.

Figure 2-8(a) plots the energy per clock cycle ($E_{act/cyc} = P_{act} * T_{cyc}$) of median filters with one comparator shared between 1 to 16 PEs. This is simulated at 1V for post-layout designs annotated with accurate switching activity and wiring parasitics. It is shown that a filter with more comparator sharing has fewer logic gates and interconnect capacitance, resulting in lower energy per *clock cycle*. Figure 2-8(b) plots the leakage power at various degress of comparator sharing. Although the leakage power decreases with more sharing, it eventually levels off when other components that cannot be shared begin to dominate the leakage power.

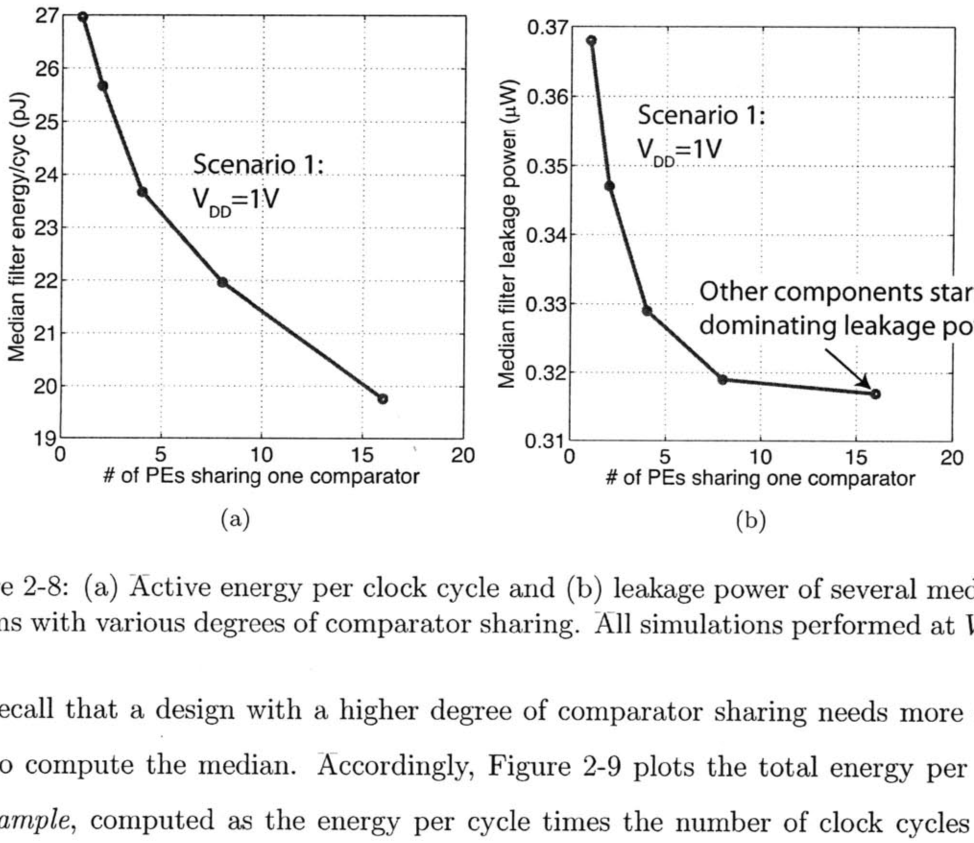


Figure 2-8: (a) Active energy per clock cycle and (b) leakage power of several median filter designs with various degrees of comparator sharing. All simulations performed at V_{DD}=1V.

Recall that a design with a higher degree of comparator sharing needs more clock cycles to compute the median. Accordingly, Figure 2-9 plots the total energy per *one output sample*, computed as the energy per cycle times the number of clock cycles required ($E_{act/cyc} * N_{cyc}$). This illustrates the trade-off between a median filter with no comparator sharing (i.e. higher $E_{act/cyc}$ and lower N_{cyc}), versus a design with more sharing (i.e. small

$E_{act/cyc}$ and large N_{cyc}).

In Figure 2-9, the term $E_{act/cyc}N_{cyc}$ gives the y-intercept of the curves while P_{leak} gives the slope. Even though the design with no sharing has larger $E_{act/cyc}$, it consumes lower active energy overall due to a small N_{cyc}. Therefore, when the active energy component dominates – when the time between consecutive input samples T_{acc} is small – the design with no sharing consumes the least energy At larger T_{acc}, increased leakage starts to dominate, and a design with more sharing consumes lower energy.

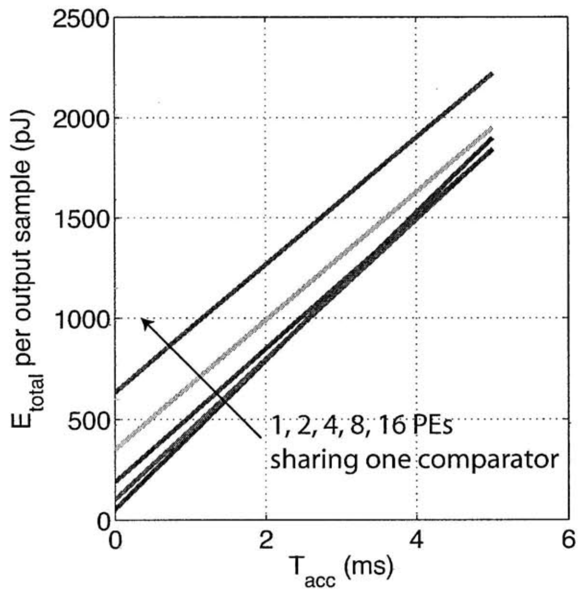


Figure 2-9: Total energy to compute *one output sample* versus T_{acc} (time between input samples). This shows the total energy for several median filter designs, each with a different number of PEs sharing one comparator.

Scenario 2: Independent power supply

Figure 2-10 illustrates the power consumption over time of the median filter in scenario 2, where the filter can be power-gated during idle periods and its V_{DD} can be set independently from the rest of the system. The variables are defined as follows:

- P_{act}, N_{cyc}, T_{acc}: same definition as in scenario 1.
- T_{cyc}: the critical path delay of the median filter at the V_{DD} where it operates.
- P_{leak}: the leakage power of the median filter when it is not power gated.
- T_{sleep}: the time over which the median filter is power gated in between consecutive

input samples. Note that the filter is power gated only if T_{sleep} is longer than the break-even time as defined in Section 3.3.2.

- P_{sleep}: the leakage power of the filter when it is power gated.
- $E_{pg-overhead}$: the energy overhead associated with power gating (Section 3.3.2).

The total energy per output sample in the scenario of Figure 2-10 is given by

$$E_{total} = P_{act}T_{cyc}N_{cyc} + P_{leak}T_{cyc}N_{cyc} + P_{sleep}T_{sleep} + E_{pg\_overhead} \tag{2.2}$$

It is possible to buffer several input samples before powering on the median filter to process them. The advantage of this is that $E_{pg\_overhead}$ can be amortized over several input samples. We do not consider buffering here, but it can be included easily in the analysis.

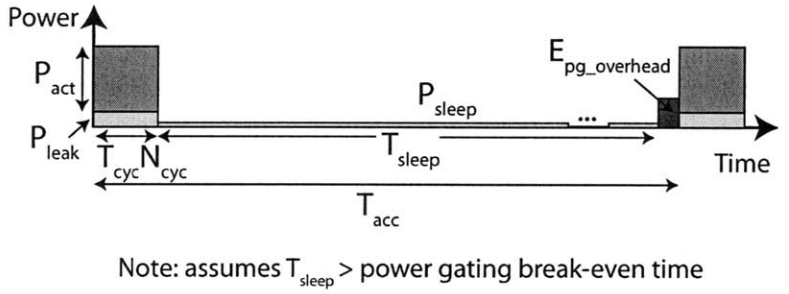


Figure 2-10: Power profile with power gating.

We now consider the effect of scaling V_{DD}. First, we constrain all designs to run at the same throughput. The design with no comparator sharing requires fewer clock cycles to compute the median versus a design with comparator sharing. As a result, the former can run at a lower frequency, which implies that it can operate at a lower V_{DD}.

Figure 2-11(a) plots $E_{act/cyc}$ versus degree of comparator sharing when V_{DD} is lowered so that all designs have constant throughput. Here, the design without sharing has the lowest active energy due to quadratic savings from V_{DD} scaling. The total energy is shown in Figure 2-11(b). When T_{acc} is small (less than the break-even time[2]), the filter is not power gated and consumes leakage power equal to P_{leak}. When T_{acc} is large, the design is power gated, after which it consumes negligible P_{sleep}. Figure 2-11(b) shows that the difference in leakage energy between designs is negligible compared to the difference in active energy arising from V_{DD} scaling. In this scenario, the median filter without any sharing achieves

[2]More precisely, the design is not power gated when T_{sleep} is less than the break-even time.

the lowest energy per operation.

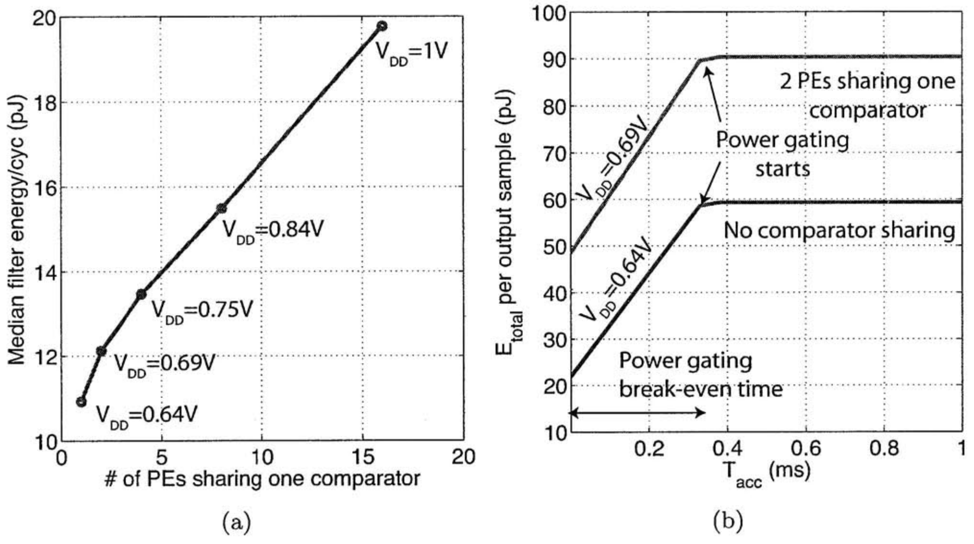


Figure 2-11: (a) Energy per *clock cycle* at different degrees of comparator sharing. V_{DD} is set so that all designs have constant throughput. (b) Total energy per *output sample*. When T_{acc} is large, the design is power gated, after which it consumes negligible P_{sleep}.

The above analysis illustrates that the system and application characteristics set the relative importance of leakage and active energy components, which in turn influence the accelerator architecture if energy minimization is the goal. The biomedical processor does not allow the median filter V_{DD} to be set independently from the rest of the system. In this case, Figure 2-9 showed that total energy per output sample is similar for a filter without comparator sharing versus a design where four PEs share a comparator. Consequently, the implemented median accelerator adopts the latter design to save area.

2.3 CORDIC Engine

As noted in Section 2.1, biomedical applications employ a range of mathematical functions that are not natively supported in fixed point processors. On a typical microcontroller, these functions are emulated in software by the compiler. However, one emulated operation typically requires several thousand cycles to complete. Fortunately, the CORDIC algorithm allows efficient computation of these functions at relatively low hardware cost, and in fact, CORDIC has been widely used in pocket calculators.

CORDIC, which stands for Coordinate Rotation Digital Computer, was first proposed in 1959 as an algorithm to compute trigonometric functions in airplane navigation systems

[51]. It was later generalized in 1971 to compute other functions such as multiplication, division, exponentials, and square roots [52]. A comprehensive survey of the developments in CORDIC theory and architectures can be found in [53]. This section first describes the CORDIC algorithm and the classic hardware architecture. Next, we discuss several limitations of the classic approach, our proposed improvements, and their impact.

2.3.1 CORDIC Algorithm

To aid understanding of the hardware architecture and its limitations, we first give an overview of the conventional CORDIC algorithm developed by [51] and [52]. For simplicity we omit some details here, but a good description of the algorithm can be found in [54]. The CORDIC algorithm computes trigonometric functions by rotating a vector in a successive approximation manner in the circular coordinate system. This can be further extended to the linear and hyperbolic coordinate systems as proposed in [52]. The key strength of this approach is that it allows computation of some mathematical functions in very few clock cycles and with low hardware cost, which makes it amenable to low power, flexible processors such as the biomedical platform in this work.

Figure 2-12 illustrates how the sine and cosine of an angle α can be computed with CORDIC. We begin with a unit vector v_0 parallel to the x-axis, then rotate it by a set of elementary angles θ_i until it makes angle α with the x-axis. Cosine and sine are then simply the x- and y-components of the resulting vector.

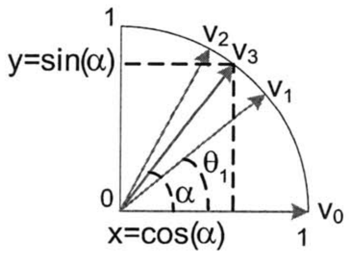


Figure 2-12: Illustration of the basic principle behind CORDIC – a unit vector v_0 is rotated until it makes angle α with the x-axis, thus providing $\sin(\alpha)$ and $\cos(\alpha)$.

Mathematically, rotating a vector $v_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ is equivalent to multiplying it by a rotation matrix. By restricting the rotation angle in each iteration to a satisfy a specific property [54], the rotation (except for a scaling factor) can be implemented efficiently by shifting and addition. The necessary scaling factors can be aggregated over all iterations and applied at the beginning or end of the iterations.

In each iteration, the vector is rotated clockwise or counterclockwise so that its angle with the x-axis approaches the desired quantity. To determine the direction of rotation at each step, an angle accumulator is first initialized with the desired angle (α). The amount of rotation at every step is then subtracted from the accumulator, and the next rotation is taken in a direction that decreases the magnitude of the angle accumulator.

The above summarizes the rotation mode of operation of CORDIC. Reversing this process gives rise to the vectoring mode of operation, where we start with a vector at an angle with the x-axis, and rotate it until it becomes parallel to the x-axis. At the conclusion, y should be close to 0, x is a scaled magnitude of the original vector, and z is the angle formed by the original vector with the x-axis.

The operations in the circular coordinate system can be further generalized to the linear and hyperbolic coordinate systems [52], thus allowing CORDIC to support a variety of functions. Equation 2.3–Equation 2.5 describe how to update the x, y components of the vector and the angle accumulator (z) after each iteration.

$$x_{i+1} = K_{mi}(x_i - m y_i d_i 2^{-i}) \tag{2.3}$$

$$y_{i+1} = K_{mi}(y_i + x_i d_i 2^{-i}) \tag{2.4}$$

$$z_{i+1} = z_i - d_i \gamma_i \tag{2.5}$$

where:

- m =1, 0, and -1 for rotations in the circular, linear, and hyperbolic coordinate systems respectively
- γ_i is set to $\arctan(2^{-i})$, 2^{-i}, and $\text{arctanh}(2^{-i})$ for the circular, linear, and hyperbolic systems respectively
- $d_i = \pm 1$ indicates the direction of rotation
- K_{mi} is a set of scaling factors as described in [54]

Table 2.3 summarizes the supported functions in six different modes of operation of the unified algorithm, as well as the required initialization, results, and any post-processing involved. Although not specifically supported in our processor, the CORDIC algorithm can be extended to compute the inverse of certain functions, for instance arcsin(θ) and arccos(θ). The procedure for doing so is detailed in [54].

Table 2.3: Functions, inputs, outputs, and post-processing in six modes of operation. All modes are supported in the CORDIC accelerator except linear-rotation (since a hardware multiplier is already available).

Mode of operation	Supported Functions	Inputs	Outputs	Post-Processing
Circular-rotation	$\cos(z_0)$ $\sin(z_0)$	$x_0 = 1/K_{nc}$ $y_0 = 0$ $z_0 = a$	$x_n = \cos(z_0)$ $y_n = \sin(z_0)$ $z_n = 0$	N/A
Circular-vectoring	$\arctan(y_0/x_0)$ $K_{nc}\sqrt{x_0^2 + y_0^2}$	$x_0 = a$ $y_0 = b$ $z_0 = 0$	$x_n = K_{nc}\sqrt{x_0^2 + y_0^2}$ $y_n = 0$ $z_n = \arctan(y_0/x_0)$	Scale x_n by $1/K_{nc}$
Linear-rotation	$x_0 z_0$	$x_0 = a$ $y_0 = 0$ $z_0 = b$	$x_n = x_0$ $y_n = x_0 z_0$ $z_n = 0$	N/A
Linear-vectoring	y_0/x_0	$x_0 = a$ $y_0 = b$ $z_0 = 0$	$x_n = x_0$ $y_n = 0$ $z_n = y_0/x_0$	N/A
Hyperbolic-rotation	$\cosh(z_0)$ $\sinh(z_0)$	$x_0 = 1/K_{nh}$ $y_0 = 0$ $z_0 = a$	$x_n = \cosh(z_0)$ $y_n = \sinh(z_0)$ $z_n = 0$	$e^{z_0} =$ $\sinh(z_0) + \cosh(z_0)$
Hyperbolic-vectoring	$\sqrt{a}$	$x_0 = a + 1/4$ $y_0 = a - 1/4$ $z_0 = 0$	$x_n = K_{nh}\sqrt{a}$ $y_n = 0,$ $z_n = \text{arctanh}(y_0/x_0)$	$\sqrt{a} =$ x_n/K_{nh}
Hyperbolic-vectoring	$\ln(a)$	$x_0 = a + 1$ $y_0 = a - 1$ $z_0 = 0$	$x_n = K_{nh}\sqrt{x_0^2 - y_0^2}$ $y_n = 0,$ $z_n = \text{arctanh}(y_0/x_0)$	$\ln(a) =$ $2\text{arctanh}(y_0/x_0)$

In Table 2.3, K_{nc} and K_{nh} refer to the aggregated scaling factor in the circular and hyperbolic coordinate systems respectively, and are given by

$$K_{nc} = \prod_i \sqrt{1 + 2^{-2i}} \tag{2.6}$$

$$K_{nh} = \prod_i \sqrt{1 - 2^{-2i}} \tag{2.7}$$

2.3.2 CORDIC Architecture

A direct mapping of Equation 2.3–Equation 2.5 to hardware components gives rise to the classic iterative CORDIC architecture pictured in Figure 2-13. It is apparent that this design has low hardware cost; x_i and y_i are computed with two adders and barrel shifters. This architecture provides one bit of the result per clock cycle. The functions $\arctan(2^{-i})$, 2^{-i}, and $\operatorname{arctanh}(2^{-i})$ for integer values of i, necessary for computing z_i, are stored in lookup tables. In several modes of operation the CORDIC algorithm introduces an extra scaling factor as noted in Table 2.3. To eliminate the need for the user to consider this extra scaling, a multiplier is employed to remove the scaling factor from the CORDIC output.

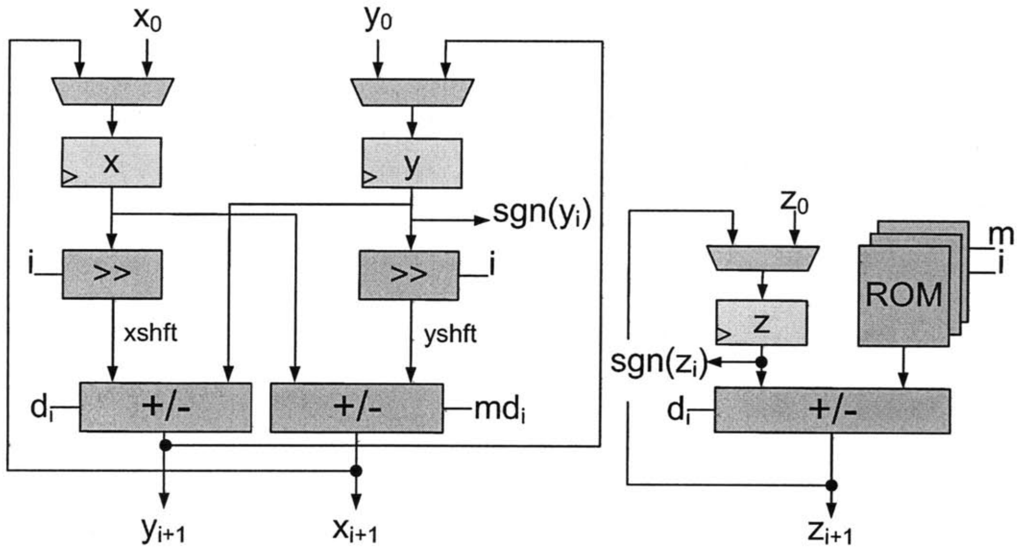


Figure 2-13: Basic iterative hardware architecture for CORDIC.

2.3.3 Proposed Hardware Modifications

The CORDIC engine in our processing platform is based on the iterative architecture in Figure 2-13 but with several proposed enhancements. Before detailing the enhancements, we first quantify the speed advantage of the CORDIC architecture over software emulation on the CPU. Table 2.4 compares the number of clock cycles needed to compute various functions on the CPU versus the accelerator, including the overhead of transferring data into and out of the accelerator. This illustrates that CORDIC is a highly efficient and versatile architecture for supporting such functions.

It is important to note that the conventional CORDIC algorithm suffers from several limitations: the algorithm introduces numerical inaccuracy and converges to the correct result for a limited range of inputs. Accordingly, we propose architectural enhancements to mitigate these effects.

Table 2.4: Number of clock cycles needed to compute various functions on the MSP430 CPU versus the CORDIC accelerator.

Function	CPU + Multiplier	Accelerator
$\cos(\theta)$	4482	50
$\arctan(\theta)$	1344	50
$\sqrt{x^2+y^2}$	7716	50
e^x	4288	88
$\ln(x)$	4214	59

Limited Convergence Range

The limited convergence range of CORDIC stems from the fact that the angle accumulator z can only be increased/decreased by values in a finite-size lookup table. When unaddressed, this severely limits the usefulness of the CORDIC accelerator in biomedical applications. Intuitively, in order for z to be driven towards zero in rotation mode, the initial input value z_0 is restricted by the sum of all entries in the lookup table. More precisely, the convergence theorem in [52] sets the bounds on z_0 for which CORDIC will converge to the correct value. In circular-rotation mode, the bound is

$$|z_0| \leq \arctan(2^{-N})) + \sum_{i=1}^{N} \arctan(2^{-i}) \tag{2.8}$$

Similar arguments and bounds apply to other modes of operation.

In the CORDIC engine of this work, the limited convergence range is addressed using different approaches in each coordinate system. In the circular system, conventional CORDIC converges for angles in quadrants I and IV. Angles in quadrants II and III are handled by the well-known method of performing an initial rotation of $\pm\frac{\pi}{2}$. In the linear system, we are interested primarily in the vectoring mode of operation which computes y/x, for which CORDIC converges when $|y_0/x_0| \leq 1$. Therefore, we support integer division by computing $1/x_0$ then multiplying the result by y_0 with the internal CORDIC multiplier.

Extending the convergence range in the hyperbolic coordinate system has received much less attention since many CORDIC designs operate only in the circular system. Further, while inputs in the circular-rotation mode are restricted[3] to the range $-\pi \leq z \leq \pi$, there

[3]Similarly, there is no convergence issue in circular-vectoring mode because $\arctan(y_0/x_0)$ lies within $\pm\pi/2$.

are no such bounds on the inputs of hyperbolic functions. One approach to address the issue involves reducing the input argument in order to place it within the convergence range of CORDIC [55, 56, 57]. In particular, the algorithm proposed by [55] can be integrated into the conventional CORDIC datapath and is thus more suitable for our CORDIC engine. Therefore, we propose architectural enhancements to realize this algorithm as part of the iterative datapath (Figure 2-13).

The conventional CORDIC converges in the hyperbolic-rotation mode when the input z_0 is

$$|z_0| \leq \operatorname{arctanh}(2^{-N}) + \sum_{i=1}^{N} \operatorname{arctanh}(2^{-i}). \tag{2.9}$$

For large N, the sum approaches 1.1182. This is a very limited convergence range, and it would be highly desirable to extend this range for general-purpose computation.

A natural step towards increasing the convergence range would involve including extra iterations with non-positive index values ($i \leq 0$). However, $\operatorname{arctanh}(2^{-i})$ is complex-valued for negative i. The authors of [55] proposed another sequence that can extend the convergence range quickly in a few iterations. Using this new sequence in $M+1$ extra iterations at the beginning, the input range is given by

$$|z_0| \leq \sum_{i=-M}^{0} \operatorname{arctanh}(1 - 2^{-2^{-i+1}})) + \operatorname{arctanh}(2^{-N}) + \sum_{i=1}^{N} \operatorname{arctanh}(2^{-i}). \tag{2.10}$$

Table 2.5 lists the maximum allowable $|z_0|$ for different values of M. In our CORDIC engine, we support M up to 3 since this provides sufficient input range for the functions of interest (e^x, $\sqrt{x}$, $\ln x$) in a 16-bit fixed point processor, as will be detailed later.

Table 2.5: Maximum allowable z_0 for different values of M. The number of extra iterations required is $M+1$.

M	z_{max}
0	2.091
1	3.808
2	6.926
3	12.818
4	24.255

The proposed architectural enhancements to realize this algorithm in the iterative CORDIC engine are as follows. In this work since we implement the algorithm in a fixed-point engine and must address two main design issues not considered in the theoretical work [55]. Specifically, the internal datapath values have a large dynamic range due to the extra iterations. This, in turn, significantly increases the quantization error in the computed results. In the following, we provide details on how this is addressed in the proposed CORDIC engine.

Although the engine supports M up to 3 (performing up to 4 extra iterations), doing so for every input regardless of its magnitude would significantly compromise numerical accuracy. This is because in the $M = 3$ iteration, more integer bits (out of a maximum of 15) must be allocated to the z register in order to accommodate the larger values involved. If the $M = 3$ iteration was not actually necessary because the input z_0 is small, more bits could have been allocated to the fractional part of z_0 to improve numerical accuracy. For this reason, M and the number of extra iterations are not set statically, but are determined in real-time for each input.

We determine M by requiring the user to give a 16-bit signed representation of z_0 and the number of integer bits in the representation. As will be addressed in Section 6.2.2, this can be done easily in software by disassembling a floating point datatype into its constituent parts. z_0 is then compared with the values in Table 2.6, and the corresponding M is selected. The ranges in Table 2.6 differ slightly from those of Table 2.5 to ease implementation. In hardware, comparison of z_0 with these ranges can reuse the add/subtract units in the conventional CORDIC datapath, hence no extra arithmetic units are required but the total execution time is lengthened by several clock cycles.

Table 2.6: Value of M selected in the CORDIC engine for a given value of z_0

Range of z_0	M
$\lvert z_0 \rvert < 1$	No extra iterations necessary
$1 \leq \lvert z_0 \rvert \leq 2$	0
$2 \leq \lvert z_0 \rvert \leq 3.75$	1
$3.75 \leq \lvert z_0 \rvert \leq 6.875$	2
$6.875 \leq \lvert z_0 \rvert$	3

The resulting CORDIC architecture is shown in Figure 2-14. Compared to conventional CORDIC, additions to extend the input range include:

- a shifter to shift values from the lookup tables to the appropriate position before adding to z_i

- multiplexers before barrel shifters and adders for comparing z_0 against the ranges in Table 2.6
- the hard-coded constants from Table 2.6

With these changes, we significantly extend the convergence range of CORDIC in the hyperbolic coordinate system. Table 2.7 compares the convergence range of conventional CORDIC with that of our implementation. The range has now been extended by approximately a factor of 2^{14} to cover the full range required in 16-bit fixed point processing.

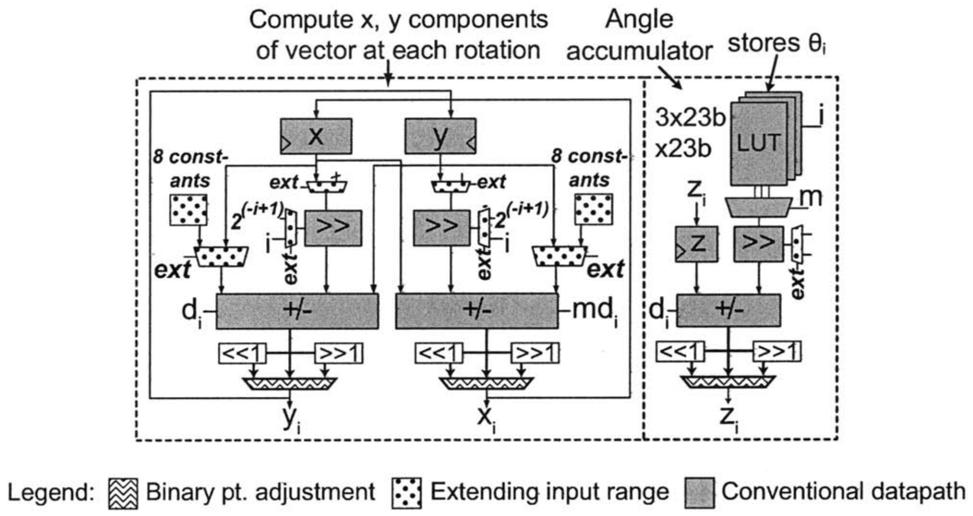


Figure 2-14: Proposed architecture to extend the convergence range (components with dotted pattern) and reduce quantization error (zigzag pattern). Due to reuse of existing datapath blocks, only limited additional components are required.

Table 2.7: Convergence range of conventional CORDIC and our proposed implementation. The range has been extended to cover the full range required in a 16-bit fixed point processor.

Operation	Conventional CORDIC	Proposed Enhancements
$\sqrt{x}$	$0.0267 \leq x \leq 2.339$	$0 \leq x \leq 2^{15} - 1$
$\ln x$	$0.1068 \leq x \leq 9.360$	$2^{-15} \leq x \leq 2^{15} - 1$
e^x	$-1.118 \leq x \leq 1.118$	$-10.39 \leq x \leq 10.39$ ($e^{10.39} \approx 2^{15} - 1$)

Quantization Error

The CORDIC algorithm contains two main sources of quantization error. Since the desired rotation angle is approximated as the sum of a finite number of elementary angles stored in the lookup table, an exact representation may not be possible. This will be referred to as the angle approximation error. Furthermore, in a fixed point processor, rounding errors are inevitable. In this section we investigate several architectural techniques to improve the numerical accuracy of the CORDIC engine.

Due to quantization errors, the effective number of bits provided by CORDIC is less than the datapath width. Thus a straightforward way to improve numerical accuracy is to increase the number of iterations to decrease the angle approximation error, and to widen the CORDIC datapath to reduce the rounding error. In [58], upper bounds on the total quantization error were derived for a conventional CORDIC design. However, due to some architectural changes detailed below, deriving similar bounds is difficult for our CORDIC engine, but this can be investigated in future work. Instead we perform extensive simulations, with a fixed-point MATLAB model which exactly mirrors the hardware design, to characterize the quantization error in the accelerator. The results are as follows.

Figure 2-15 shows the error introduced by CORDIC while computing $\sin(z_0)$ at different datapath widths. The y-axis plots the relative error defined as $\left|\frac{\text{CORDIC output}-\text{ideal value}}{\text{ideal value}}\right|$. The ideal value is quantized to 16-bits to decouple the inaccuracies introduced by CORDIC from the error inherent to representing a number in 16-bit fixed point format. As seen here, widening the datapath significantly reduces the maximum error. Since the quantization error varies with input, we take the root mean square (RMS) of the values in Figure 2-15 as an overall error metric. The trade-off between improving accuracy and increasing energy per cycle is illustrated by simulated results in Figure 2-16. From this we see that the RMS error decreases exponentially with a linear increase in energy. In this design we employ a 24-bit datapath to achieve good accuracy in the hyperbolic coordinate system which is the most susceptible to quantization effects.

The second approach to improving accuracy involves adjusting the binary point dynamically to maximize the number of fractional bits used throughout the computation. Arithmetic overflow is a particular concern in fixed-point computation, and overflow is handled in the CORDIC engine by dedicating 1 bit in x, y, and z as an overflow bit, as illustrated in Figure 2-17(a). An overflow occurs when the overflow bit is different from the sign bit, and this condition is checked before the registers are updated. If either x_i or y_i overflows, both variables are shifted towards the right by 1 position before being registered, since the binary points of x and y must be at the same position. Overflow for z is checked independently. Two counters store the number of times the binary point was shifted to the

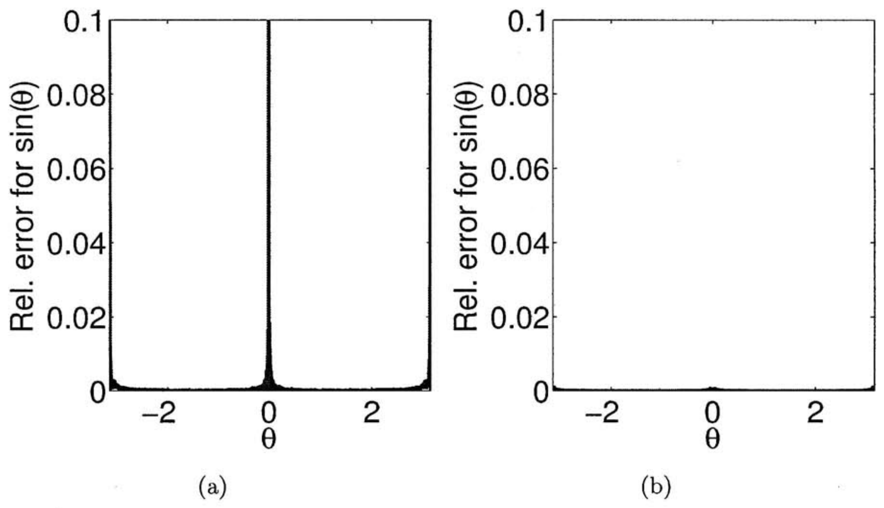


Figure 2-15: Quantization error of CORDIC engine with (a) 16-bit and (b) 24-bit datapath while computing $\sin(\theta)$.

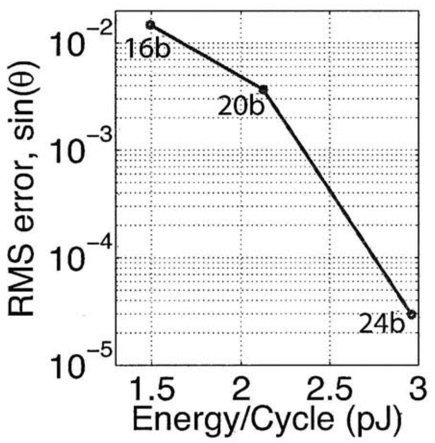


Figure 2-16: RMS quantization error (calculated over θ in $[-\pi, \pi]$) in the modified CORDIC engine. This is plotted versus simulated energy per cycle at different datapath widths.

right so that the end result can be interpreted accordingly.

Similarly, the binary point can be adjusted to increase the number of significant binary bits whenever possible, to help reduce rounding errors. When the sign, overflow, and most significant bits of a variable are the same (see Figure 2-17(b)), the MSB is redundant, and therefore the variable can be shifted towards the left by one position without losing information. x and y are adjusted if both satisfy this condition, while z is adjusted independently.

Recall that the algorithm to extend the input range also increases the dynamic range of the intermediate datapath values, and thus increases the quantization error. Binary point

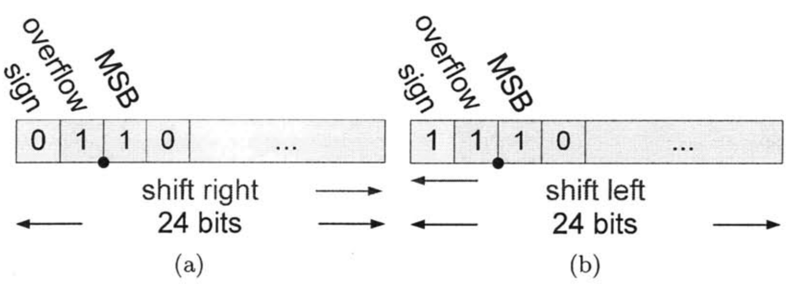


Figure 2-17: (a) Overflow bit in the x, y, and z registers. If an overflow occurs, register contents are shifted one position towards the right. (b) When the leftmost three bits are equal, the MSB is redundant, and the variable is shifted one position towards the left.

adjustment is particularly effective in reducing the quantization error in these scenarios. To illustrate, Figure 2-18(a) plots the relative error versus different input values for $\sqrt{x}$ without binary point adjustment. In Figure 2-18(b), the relative error has been significantly decreased with binary point adjustment.

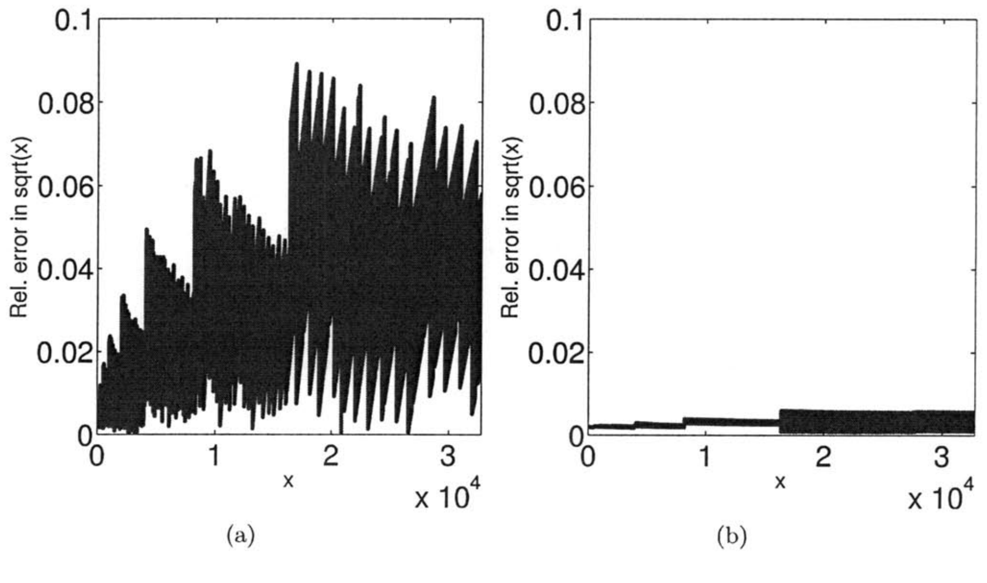


Figure 2-18: Relative quantization error in $\sqrt{x}$ that is introduced by CORDIC (a) without and (b) with binary point adjustment. The maximum error has been decreased in (b).

Figure 2-19 and Figure 2-20 illustrate the quantization error profile of the proposed CORDIC engine in the circular-rotation ($\sin(x)$), linear-vectoring (y/z) and the hyperbolic-rotation ($\sinh(x)$, e^x) modes of operation. It can be seen that the quantization error is difficult to predict in general.

Table 2.8 summarizes improvements in the RMS error attributed to the binary point adjustment technique under operation in all three coordinate systems. In the circular-

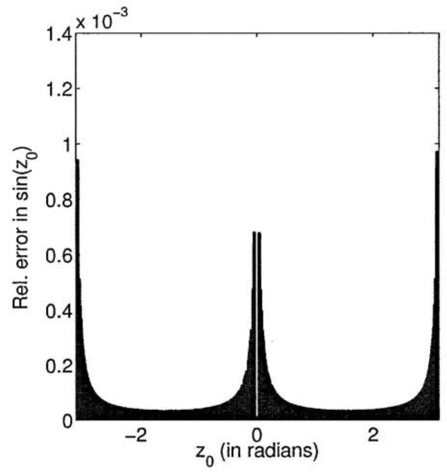


Figure 2-19: Relative numerical error in the proposed CORDIC engine in the circular-rotation mode of operation.

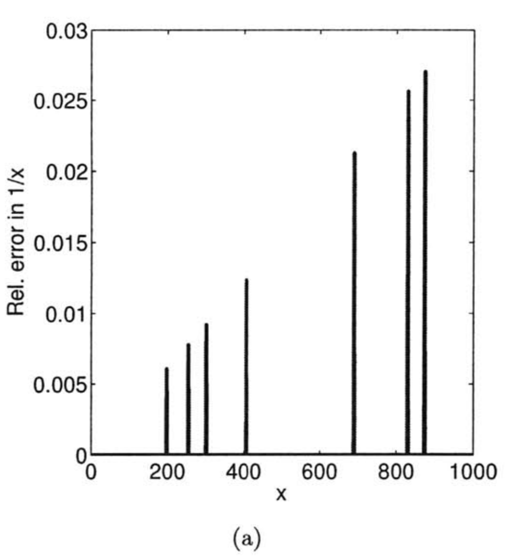
rotation mode, the technique results in approximately 2× improvement in RMS error. It is interesting to note that the linear-vectoring mode shows no benefit since overflow/underflow occur very infrequently in this mode. Conversely, the hyperbolic-rotation mode benefits greatly since the intermediate results can grow quickly in this case.

Table 2.8: RMS error introduced by CORDIC with and without binary point adjustment.

Mode of operation	Without adjustment	With adjustment
Circular-rotation	5.75×10^{-5}	2.95×10^{-5}
Linear-vectoring	0.00147	0.00147
Hyperbolic-rotation	0.8136	0.002

The last method to improve accuracy pertains to exponentials, which are computed conventionally as $e^z = \sinh(z) + \cosh(z)$ [54]. Because $\sinh(z)$ and $\cosh(z)$ are large in magnitude but opposite in sign when z is negative, the resulting e^z calculation is grossly inaccurate. This is reflected by the large errors for negative x in Figure 2-21(a).

In this CORDIC engine, the issue is addressed by first computing $e^{|z|}$ for $z < 0$, then leveraging the linear-vectoring mode of operation to invert the result. This is accomplished at low hardware cost of several multiplexers to obtain and direct $|z|$ appropriately during startup, as well as simple logic to initiate the $1/e^{|z|}$ computation. Moreover, the numerical error is significantly reduced as shown in Figure 2-21(b). Although the relative error for $x \approx -10$ seems large, it is important to note that $e^{-10} \approx 4.54 \times 10^{-5}$ which is between the two smallest positive signed 16-bit numbers, 3.053×10^{-5} and 6.103×10^{-5}. In this region,



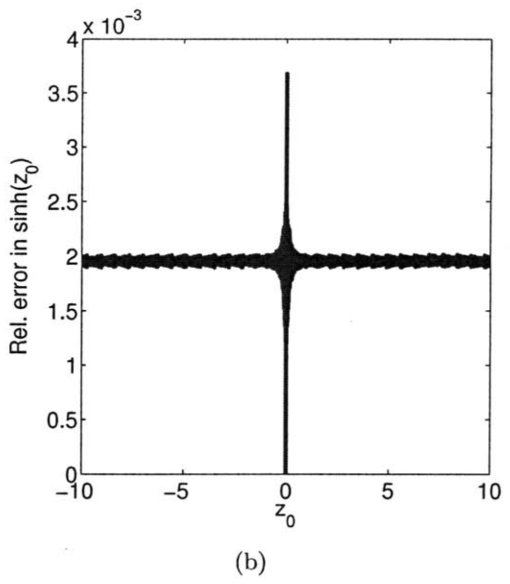
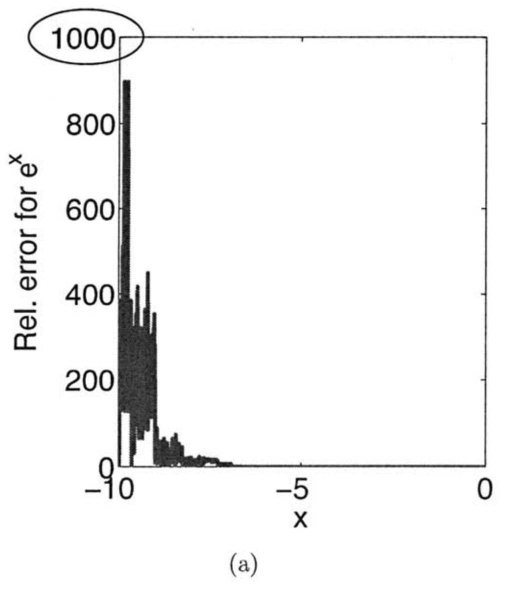


Figure 2-20: Relative numerical error of the proposed CORDIC engine in (a) linear-vectoring and (b) hyperbolic-rotation modes of operation.

a small quantization error of 1 LSB can easily lead to a large relative error.

Figure 2-22 summarizes the impact of both enhanced convergence range and improved accuracy on e^x. The input convergence range increased by 9.3× and accuracy has been substantially improved for $x < 0$.



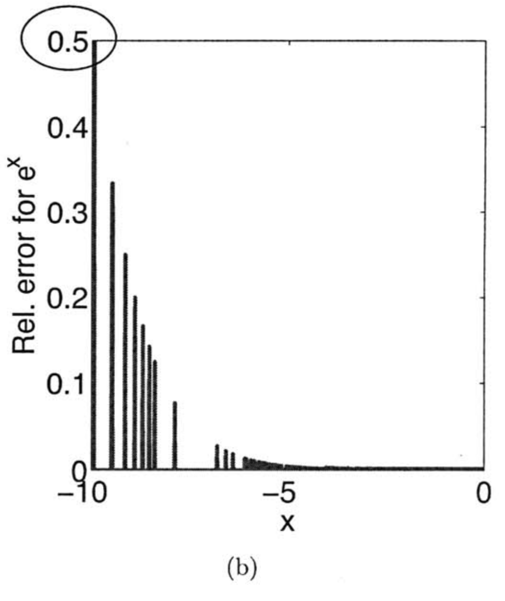


Figure 2-21: Relative numerical error in e^x introduced by (a) conventional approach and (b) proposed approach.

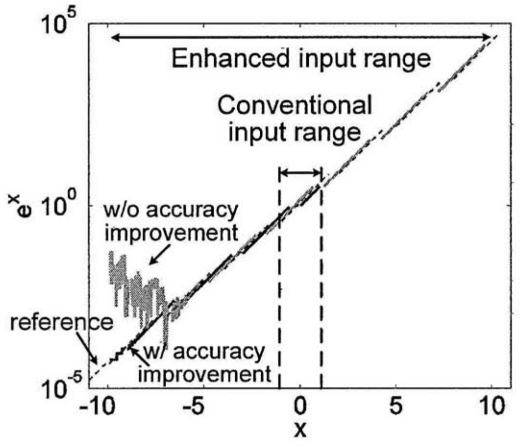


Figure 2-22: Input convergence range and accuracy of conventional CORDIC and the proposed improvements. Dotted line shows e^x value in double precision, to be compared against the CORDIC outputs.

2.4 Fast Fourier Transform

The Fast Fourier Transform is an efficient algorithm for computing the Discrete Fourier Transform (DFT), which is important to many applications in signal processing, for example in implementing digital filters and in spectral analysis. In this section we will propose a control scheme to reduce the switching activity in the FFT datapath. This decreases the datapath power by 50% and the overall FFT power by 29%. As with the CORDIC engine, we also discuss ways to extend the classic architecture to support more use cases at low hardware cost.

2.4.1 Architecture Overview

The FFT accelerator utilizes a serial radix-2 architecture shown in Figure 2-23. It contains a datapath to implement one butterfly computation which consists of one complex multiplication and two complex additions/subtractions as detailed in [59]. An N-point FFT requires $N \log_2(N)/2$ butterflies; in the accelerator, the control logic is responsible for sequencing the order of butterfly computations. The twiddle ROM stores twiddle factors which are coefficients needed in the FFT computation. The input data and intermediate results are stored in a custom voltage-scalable SRAM, allowing the FFT accelerator to function down to 0.5V. The FFT SRAM employs the same bit-cell and peripheral circuit design as the main memory of the biomedical processor (discussed in Chapter 5).

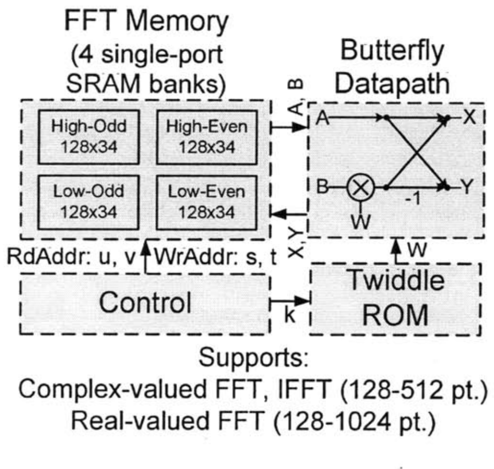


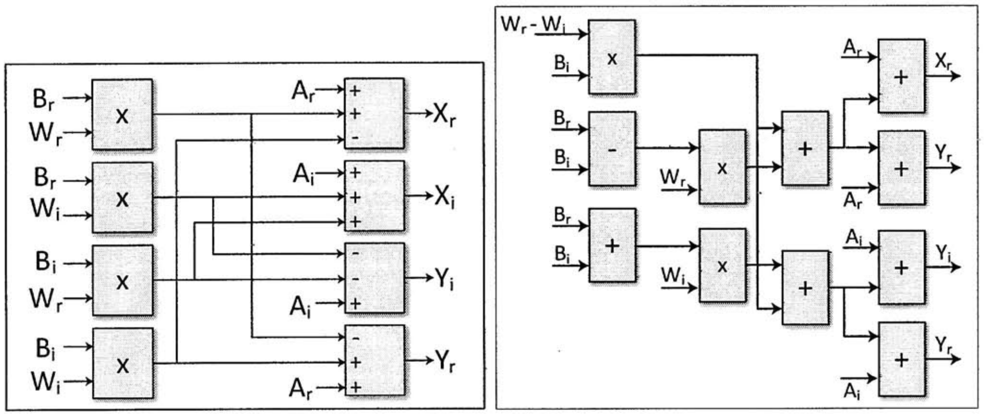
Figure 2-23: Serial radix-2 FFT architecture.

2.4.2 Low Power Optimizations

The serial radix-2 architecture is widely used and our particular implementation is based on [60] and [28]. We now propose several optimizations to reduce power and increase versatility of this basic structure.

Complex Multiplier

We first examine the complex multiplier, a key component of the butterfly datapath. In the FFT, the complex multiplier implements $(Re\{b\}Re\{w\} - Im\{b\}Im\{w\}) + j(Re\{b\}Im\{w\} + Im\{b\}Re\{w\})$. Figure 2-24(a) represents a straightforward mapping of the above equation and requires four multipliers. To decrease the number of multipliers, an alternate design shown in Figure 2-24(b) was proposed previously. Here we evaluate both designs to determine which is more suited for the FFT accelerator.



(a) (b)

Figure 2-24: (a) Butterfly datapath with 4 multipliers. (b) An alternate design with 3 multipliers.

Table 2.9 compares area, delay, and power of the two complex multipliers after synthesis. For the first set of results the multipliers were synthesized without delay constraints. In the second set, they were synthesized for a delay of 30ns, an appropriate performance for the FFT accelerator. When the delay is unconstrained, the alternate multiplier is a better choice because it occupies less area and achieves the same power consumption as the standard design. However, at the delay constraint of 30ns targeted for the FFT accelerator, the standard design is preferable. The alternate design uses larger standard cells since it contains a longer critical path, and consequently it occupies more area and consumes higher power. Because the standard design is lower power in the frequency range of interest, it was adopted for use in the butterfly datapath.

Table 2.9: Area, delay, and power estimates from synthesis of two complex multiplier designs. Synthesis performed at V_{DD}=1V.

Complex Multiplier Type	Area (μm^2)	Delay (ns)	Active Power (μW)
No delay constraint			
Standard (4 multipliers)	6301	81.1	17.18
Alternate (3 multipliers)	5302	87.4	17.08
Delay constraint = 30ns			
Standard (4 multipliers)	7679	28.32	684
Alternate (3 multipliers)	7718	29.83	824

Activity Factor Reduction

In a serial architecture, different inputs are typically time-multiplexed into a critical circuit (e.g. a datapath). Generally, if consecutive inputs are uncorrelated, time-multiplexing tends to increase the switching activity in the critical circuit. In this section we propose an approch to mitigate this in the FFT accelerator. Specifically, we present an efficient control scheme to perform butterflies with the same twiddle factor consecutively, in order to reduce the switching activity in the butterfly datapath.

Control schemes for serial radix-2 FFT [61, 62, 63, 64] have been described in previous work. The proposed control scheme differs from the cited work in that we focus on reducing the datapath switching activity, while also enabling simple single-port SRAMs to be used. Specifically, the control scheme achieves these objectives:

- In iteration i of an FFT, $i \in 0..\log_2(N)-1$, butterflies with the same twiddle factor are performed consecutively.
- On every clock cycle, two inputs are read from memory and two outputs are written back. Reads and writes should occur on different memory banks.
- The sequence should be applicable to different FFT lengths.

Since two reads and two writes are needed per clock cycle, the dedicated FFT memory is partitioned into four banks as suggested by [60] so that each bank can accommodate one access. Specifically, every memory address is assigned to one of the four banks according to its parity and most significant bit. This is because the two inputs of a butterfly always differ in parity [65] while the two inputs of the post-processing for real-valued FFT computation differ in the MSB, as will be discussed in the next section. This partitioning ensures that two reads always take place on different banks.

At the same time, the control logic is responsible for ordering the butterflies so that consecutive butterflies access different banks. For example, if the current butterfly writes outputs to the two banks with MSB=0, the next butterfly must fetch inputs from the MSB=1 banks.

We now propose the control scheme described below which satisfies the above conditions and is amenable to hardware implementation. Let a butterfly computation be represented by

$$X'[u] = X[u] + W[k]X[v] \tag{2.11}$$

$$X'[v] = X[u] - W[k]X[v] \tag{2.12}$$

where $X[u], X[v]$ are the inputs, u, v are memory addresses, and $X'[u], X'[v]$ are the outputs. $W[k]$ is the twiddle factor.

In the first $i = 0, 1, ..(n-2)$ iterations, u, v of the j^{th} butterfly differ in parity, while u, v of consecutive butterflies differ in the MSB. One method to generate u, v is:

$$m = \{j[n-2:1], 0\} \tag{2.13}$$

$$u = \{\text{j}[0],\ \text{ROL}_{n-1}(m, i)\} \tag{2.14}$$

$$v = \{\text{j}[0],\ \text{ROL}_{n-1}(m+1, i)\} \tag{2.15}$$

$$k = j \text{ with (n} - 1 - \text{i) LSBs set to 0} \tag{2.16}$$

where $j[0]$ indicates bit 0 of j, and $\{a, b\}$ denotes a concatenated with b. $\text{ROL}_{n-1}(a, b)$ involves rotating an $n-1$ bit number, a, by b bits to the left.

In the last iteration ($i = n-1$), u and v differ in their MSB so the above does not apply. Since successive u should differ in parity, u can instead be generated with an $(n-1)$-bit gray code counter, while $v = u$ with MSB set to 1. Table 2.10 gives an example of the butterfly ordering for N=16. The memory is partitioned into four banks as follows:

- Memory bank 0: addresses with even parity, MSB=0
- Memory bank 1: addresses with odd parity, MSB=0

- Memory bank 2: addresses with even parity, MSB=1
- Memory bank 3: addresses with odd parity, MSB=1

Table 2.10: Example of butterfly ordering to reduce switching activity in a 16-point FFT. Note that changes in the twiddle address are minimized within each iteration.

	j (Butterfly #)							
iteration $i = 0$								
u	0	8	2	10	4	12	6	14
memory bank of u	0	3	1	2	1	2	0	3
v	1	9	3	11	5	13	7	15
memory bank of v	1	2	0	3	0	3	1	2
twiddle address k	0	0	0	0	0	0	0	0
iteration $i = 1$								
u	0	8	4	12	1	9	5	13
memory bank of u	0	3	1	2	1	2	0	3
v	2	10	6	14	3	11	7	15
memory bank of v	1	2	0	3	0	3	1	2
twiddle address k	0	0	0	0	4	4	4	4
iteration $i = 2$								
u	0	8	1	9	2	10	3	11
memory bank of u	0	3	1	2	1	2	0	**3**
v	4	12	5	13	6	14	7	15
memory bank of v	1	2	0	3	0	3	1	2
twiddle address k	0	0	2	2	4	4	6	6
iteration $i = 3$								
u	0	1	3	2	6	7	5	4
memory bank of u	0	1	0	1	0	1	0	1
v	8	9	11	10	14	15	13	12
memory bank of v	**3**	2	3	2	3	2	3	2
twiddle address k	0	1	3	2	6	7	5	4

Note in Table 2.10 that butterflies using the same twiddle factor are performed consecutively within an iteration. Further, u and v occupy different memory banks in a butterfly, and consecutive butterflies access different banks as much as possible. The last butterfly of iteration 2 and first butterfly of iteration 3 both access memory bank 3 (shown in bold), which is unavoidable. This is handled by stalling the datapath by one clock cycle. This control scheme achieves fewer stalls than a reference FFT design without switching activity reduction.

The circuit to realize this control scheme is shown in Figure 2-25. It consists of two bit-rotators for Equation 2.14 and Equation 2.15 and a gray code counter. In general the bit-rotators need to be $log_2 N - 1$ bits wide for an N-point FFT. Therefore, to accommodate variable FFT sizes, the bit-rotators support variable bit widths as shown in the dashed box of Figure 2-25. Similarly, several gray code counters of different bit-widths are included. Figure 2-26 shows the power comparison of a reference FFT design with the proposed scheme. Both designs were synthesized and laid out, and the power consumption simulated with wiring parasitics included. It is seen that the proposed control logic consumes slightly higher power than the reference design but helps reduce the datapath power by 50%, which is a major component of the FFT power consumption. This results in an overall power decrease of 29%.

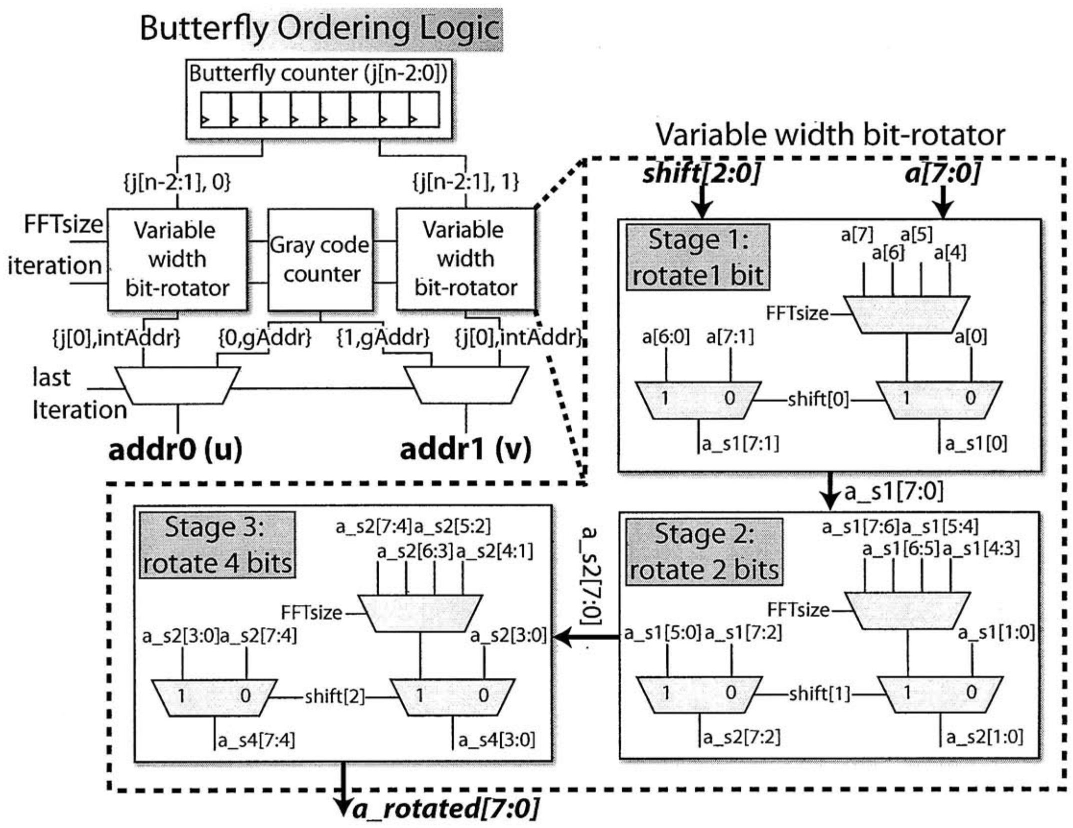


Figure 2-25: FFT control logic which reduces switching activity in the datapath.

Figure 2-27 shows the simulated waveform of the FFT using the proposed control scheme and a reference design. The switching activity at the twiddle factor input of the datapath (w_r and w_i in the waveform) is much reduced particularly in the earlier iterations. For example, in the second iteration w_r changes from 0x7FFF to 0x0000 on every cycle using a straightforward ordering scheme, while the transition only happens on one cycle in the proposed method.

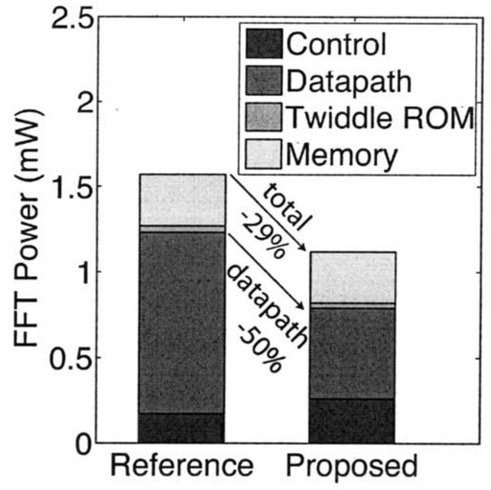


Figure 2-26: Power comparison of reference design and proposed design with switching activity reduction. Simulation performed at 1V, 10MHz for a 128-pt CVFFT.

2.4.3 Increasing Versatility

The FFT module described above supports a straightforward complex-valued FFT (CVFFT), where the inputs are assumed to be complex. However, we can make minor additions in order to leverage the basic structure for several other related computations, namely

- Real-valued FFT
- Inverse FFT
- Magnitude approximation

Real-valued FFT (RVFFT) is an optimization which allows computing the N-point FFT of a real-valued input sequence $x[n]$ via an N/2-point FFT plus a post-processing step [60]. The real-valued sequence is split up into two half-length sequences consisting of the even and odd samples ($x_e[n]$ and $x_o[n]$). We then form the complex sequence $x_e[n] + jx_o[n]$ and perform an N/2-point FFT. From this we can reconstruct the FFT of $x[n]$ in a post-processing step by taking advantage of the conjugate symmetry of $FFT\{x[n]\}$. This approach allows computing an N-point RVFFT in $2\times$ fewer clock cycles than a CVFFT of the same size. The post-processing step requires the addition of four adders to the datapath. The control overhead is negligible since the gray code counter for CVFFT control above is also applicable to the post-processing step.

Supporting the inverse FFT allows a signal to be processed in the frequency domain and then converted back into the time domain. The inverse FFT can be computed with

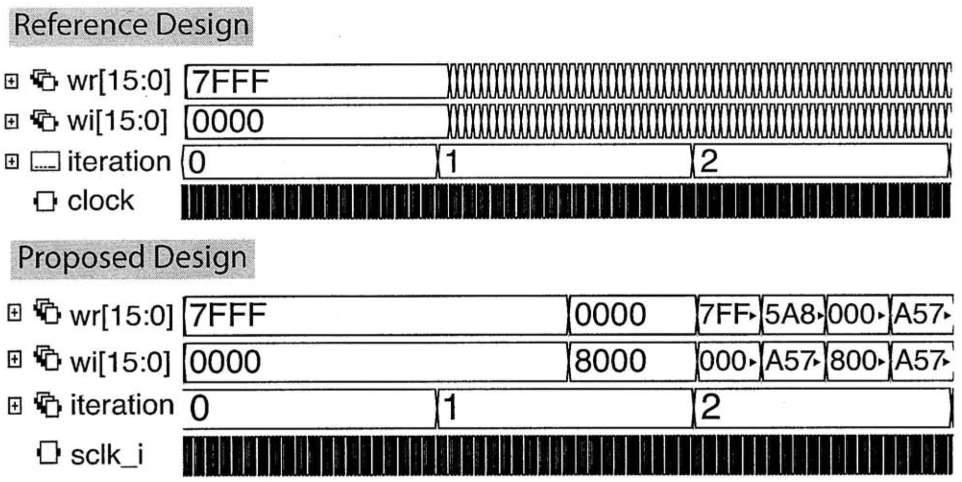


Figure 2-27: Simulated waveform of the twiddle factor w_r and w_i in the reference and proposed design. The switching activity in both signals has been substantially reduced, particularly in the early iterations when only several distinct twiddle factors are in use.

the existing hardware by swapping the real and imaginary parts of the frequency domain samples $X[k]$, performing an FFT, then swapping the real and imaginary parts of the result and scaling by $1/N$. Both swapping and scaling can be easily supported with very small hardware overhead.

Lastly, the FFT module can provide the approximated magnitude of the results, useful when the magnitude is of interest but not the phase. Again, this is achieved at low cost with the approximation $|x| \approx max(|Re\{x\}|, |Im\{x\}|) + min(|Re\{x\}|, |Im\{x\}|)/4$.

2.5 FIR Filter

As seen in Section 2.1, FIR filtering is a prevalent task in ambulatory monitoring applications. This section describes hardware architectures for FIR filtering and low power optimizations. Like the FFT module, the FIR filter has several additional features to support special cases efficiently.

2.5.1 Hardware Architectures

A N^{th} order FIR filter is described by

$$y[n] = \sum_{k=0}^{N} h[k]x[n-k] \qquad (2.17)$$

where $x[n]$ is the input sequence, $h[n]$ are the filter coefficients, and $y[n]$ is the filter output. FIR filter architectures are therefore centered around multiply-accumulate operations. A common serial FIR architecture is shown in Figure 2-28, where one multiplier and accumulator can be time-multiplexed to compute Equation 2.17 over $N + 1$ clock cycles. This architecture offers flexibility in supporting filters of variable orders, symmetric/non-symmetric filters, and other special cases. In addition, it can support high-order filters with much smaller area than a parallel implementation (e.g. [59]). We now describe low power optimizations of this serial FIR architecture.

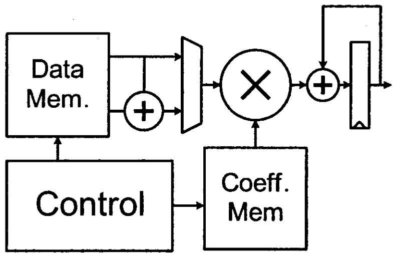


Figure 2-28: Serial FIR architecture with one multiplier and one accumulator.

2.5.2 Low Power Design

The multiply-accumulate circuit constitutes a major portion of the power and area of the FIR filter. Out of several possible multiplier designs, a Booth-Wallace multiplier was chosen for its low delay and area [50]. A Booth multiplier encodes one of the inputs with modified Booth's recoding, which reduces the number of partial products to $N/2$. It was observed in [66] that in an FIR filter, the choice between encoding the input data versus encoding the coefficients impacts the multiplier power due to different switching activity profiles. We therefore simulate the power of two FIR filters, one applying Booth encoding to the input sequence, and the other to the coefficients. Since FIR filter coefficients tend not to change dramatically from one sample to the next, we simulate power with two sets of coefficients: one randomized and one implementing a low-pass filter. The results are summarized in Table 2.11. We observe that Booth-encoding the coefficient input results in lower power in both cases.

Besides the multiply-accumulate block, the local memory to store data and coefficients is a key component of the FIR filter. Here we evaluate different approaches to implementing

Table 2.11: Power of multiply-accumulate block with Booth-encoding on the input data or coefficients.

	Multiply-Accumulate Power @ 1V, 10MHz	
Filter Coefficients	Encode Input (μW)	Encode Coefficients (μW)
32-tap Low Pass Filter	58.0	33.5
Random Coefficients	59.6	47.0

this memory. The data and coefficients are stored in two separate memories. The coefficient memory provides one word per clock cycle. However, the data memory needs to provide up to two words per cycle to handle symmetric filters, as discussed in Appendix A. Typically this type of memory would be implemented as an SRAM or a register file. An SRAM has a dense storage cell but its peripheral circuits can impose significant area overhead, particularly for the small memory sizes of interest here. Further, the need to provide two words per clock cycle would complicate the SRAM design, and consequently we employ register files for the local memories. For simplicity, the register file is synthesized from standard cells which are robust down to 0.5V; a custom register file bit-cell, like an SRAM bit-cell, would require considerable design effort to ensure low-voltage operation.

A typical synthesized register file in digital circuits use edge-sensitive flip-flops as storage elements. In this work, to save power, we employ level-sensitive latches which are approximately half as small as flip-flops and impose half the amount of clock loading. The main design issue in a latch-based register file is the possibility of a race condition, which causes data to be written into the wrong row at the end of the write cycle. In this work, this is addressed by ensuring that hold time constraints are met in the relevant signals in the register file.

The output multiplexer in the register file selects one out of 32 16-bit entries and thus constitutes a fair portion of the area and power. In this section, we simulate a design using a tri-state bus to connect all 32 entries; only the selected row actively drives the bus while all others remain high-impedance. This involves one tri-state buffer per storage bit rather than a 32:1 multiplexer tree per 32 bits. A tri-state bus requires careful design to avoid enabling two tri-state drivers simultaneously, causing excessive short-circuit current.

A latch-based local memory design with tri-stated outputs is shown in Figure 2-29. This design was realized in schematic form but was not implemented in silicon due to lack of time to manually implement the layout and characterize its timing. The 32 rows are connected by a tri-stated bus, and each bit of the bus contains a keeper to maintain the most recent value driven on the bus when no reads take place (*RDEN=0*). One advantage of this design is that it can be completely realized with standard cells, which greatly reduces design effort in physical layout and timing characterization.

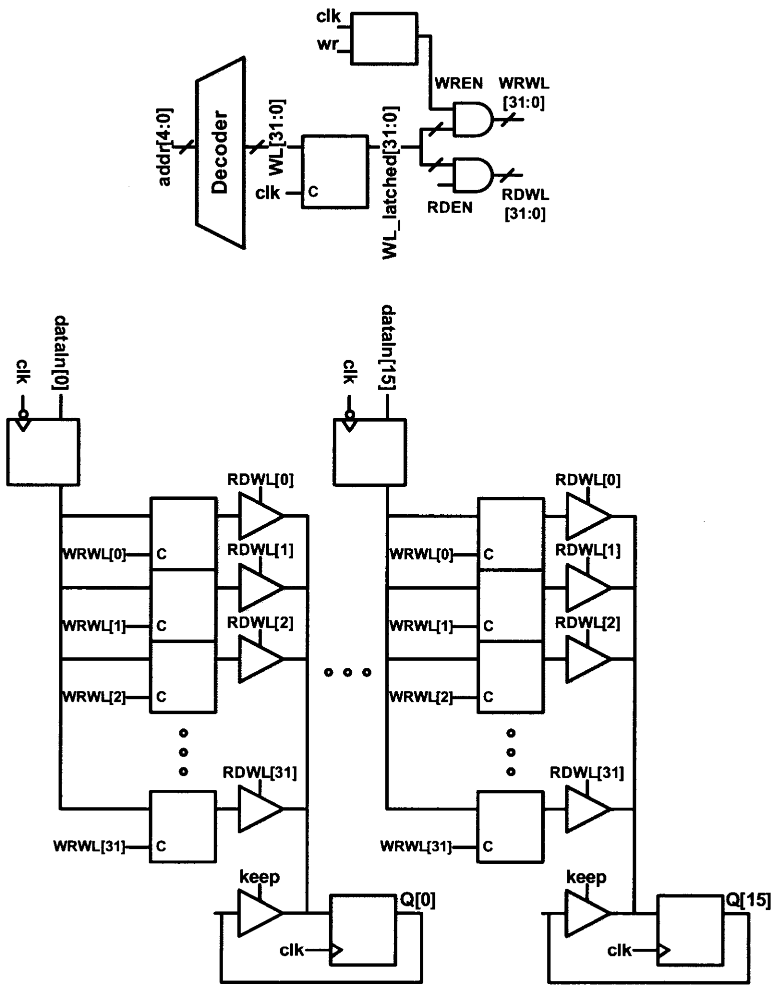


Figure 2-29: Latch-based register file design with tri-stated outputs.

Three register files – (1) a typical design synthesized from flip-flops, (2) a synthesized design with latches, and (3) a custom design with tri-stated outputs (Figure 2-29) – were simulated for power. The total FIR power and breakdown between components are shown in Figure 2-30. In the typical design, the control logic and clocking consume a significant portion of the power. Comparing designs (1) and (2), using latches in place of flip-flops reduces the memory power as well as the power consumed by the clock tree and hold time buffers. This results in a 31.1% decrease in the overall FIR power. Further, the custom-designed register file with latches and tri-stated outputs consumes 67% less power than design (1), and decreases the overall FIR power by 39.2%. Due to design time constraints as mentioned previously, the second design (synthesized with latches) was chosen for the biomedical processor.

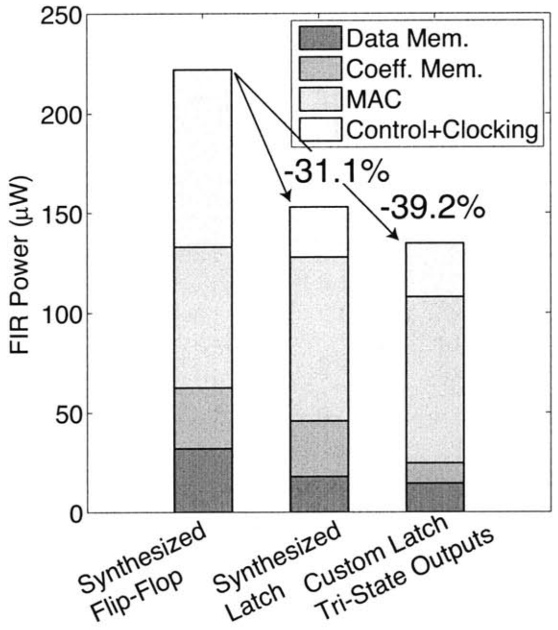


Figure 2-30: Power breakdown and comparison between three register file designs. Simulations were performed at 1V, 10MHz.

2.5.3 Cycle Count Optimizations

Since FIR filtering is widely used, it is important to leverage the basic structure shown in Figure 2-28 to support a range of usage scenarios. In this work we make several modifications to decrease the cycle count while handling special cases, and these are described in Appendix A. An important observation here is that the extra features can be realized with a small area cost. As a measure of the area overhead, Table 2.12 lists the area after synthesis of the FIR accelerator, as the features are added. The baseline design is a basic serial FIR filter without any additional features. As Table 2.12 shows, each feature only adds an area overhead of several percent, but can greatly increase the applicability of this FIR filter. For example, support of high order filters enabled energy reduction in an EEG feature extraction application as will be shown in Section 6.3.

Table 2.12: Area overhead of additional features in the FIR filter.

Features Added	Area after synthesis (in equivalent gates)	Percentage increase from above
Baseline	9061.5	N/A
Symmetric and high order filters	9687.75	6.9%
Context switching and polyphase	10164	4.9%
Multiplication by window	10206.75	0.42%

2.6 Conclusions

This chapter focused on the selection and design of accelerators for the processing platform. A survey of the literature reveals that common signal processing tasks can contribute a large portion of the total cycle count in ambulatory medical monitoring algorithms. Further, the cycle count and thus energy of these tasks can be significantly decreased through hardware acceleration. Not surprisingly, FIR filtering and FFT are widely used. The median filter and CORDIC engine are somewhat less familiar but can nonetheless support many applications – the former to remove baseline wander or noise spikes, and the latter to compute basic mathematical functions.

The median filter is based on a sorted list architecture whose hardware complexity grows linearly with window size. While a previous design utilized two comparators per entry in the window, we examine the trade-offs between area, energy, and latency of sharing a comparator amongst several entries. Simulations show that a median filter with comparator sharing has less area and switched capacitance, but its latency increases linearly with the degree of sharing. The exact energy trade-off depends on the idle time of the filter and whether its supply voltage can be changed independently from the rest of the system.

This chapter described improvements to a conventional CORDIC architecture to accommodate a range of use cases encountered in diverse algorithms. Specifically, the useful input range of the CORDIC engine can be dramatically improved with relatively few additions that fit within the conventional datapath. In the design of fixed point arithmetic blocks, it is important to consider overflow and quantization error; strategies to mitigate these were discussed in this work.

In serial architectures, an effective approach to reducing power is to decrease switching activity by reordering operations. This chapter demonstrates the concept in the FFT accelerator. The proposed butterfly ordering scheme allows the use of simple single-port SRAMs and decreases the switching activity, and thus datapath power, by 50%. The overall FFT power is reduced by up to 29%.

Finally, the FIR filter utilizes a serial architecture which, compared to a parallel design, is much more easily extensible to high-order filters, symmetric filters, and other special cases. A serial architecture commonly employs a flip-flop based local memory to store data and coefficients. We contribute here by demonstrating and characterizing a latch-based register file design, which reduced the overall FIR power by 31%. We also show that with careful design of the filter control, the accelerator can handle high-order filters which are instrumental to the EEG feature extraction application in Chapter 6. We show that these features are achieved with less than 10% area overhead compared to a bare-bones FIR filter.

Chapter 3

System Architecture: Interconnect and Power Distribution

Designing a low power and flexible system architecture is as important as the design of accelerators. Accelerators, like any other functional module, add capacitive loading on the system bus as well as leakage power. Accordingly, in this chapter we first examine the overhead of interfacing the accelerators to the rest of the processing platform. We then discuss the bus architecture, comparing the single-bus structure common in low power processors with an alternate dual-bus structure. A methodology is proposed to assign functional modules to the two buses in order to minimize the bus energy. To address leakage of the modules when not in use, this chapter concludes with a discussion of power gating implementation and trade-offs.

3.1 Accelerator Integration

In Chapter 2 we discussed the design and performance of the accelerators in isolation, but we must integrate them into the processing platform in order to leverage them in larger applications. This section addresses the overhead of interfacing the accelerators with the CPU.

The accelerators are integrated in the same way as any other memory-mapped peripheral in the platform. The CPU sets options and initializes the accelerators by writing to their registers. These register interfaces in all accelerators and peripherals reside in the same memory space as the main memory, and hence programming and retrieving results from the accelerators are functionally no different from accessing main memory. The usage of the accelerators is as follows:

- CPU configures accelerator options by writing to its register interface.
- Accelerator startrs computing, during which the CPU can perform other tasks or enter clock-gated sleep mode.
- The accelerator signals task completion.
- The CPU or Direct Memory Access (DMA) fetches the results.

Key to this usage model is the transfer of data into and out of the accelerators, which can be accomplished in the following ways.

1. *Using DMA (abbreviated as **1. DMA**)*: A channel on the DMA module can be assigned to provide inputs and/or fetch outputs from the accelerator. DMA activities take place automatically in the background without user intervention.
2. *Software-Initiated (abbreviated as **2. CPU**)*: There is no automatic notification when the accelerator finishes computing. The user software must contain explicit instructions to move each input sample to the accelerators and retrieve each output after waiting a sufficient amount of time.
3. *Using Interrupts (abbreviated as **3. ISR**)*: The accelerator automatically raises an interrupt when it needs the next input or has finished computing an output. The CPU then executes an interrupt service routine to transfer data to/from the accelerator.

These three approaches involve a different number of clock cycles and accesses to main memory, which has significant implications on the energy needed to transfer data to/from the accelerators. The cycle latency and memory accesses are determined by the MSP430 architecture (published in [16]). These values for transferring *one 16-bit word* are summarized in Table 3.1 and used in subsequent analysis, but first, we give insights into how they are derived.

1. *DMA*: DMA is active for two clock cycles per transfer of one 16-bit word (one cycle to read from the source, another to write to the destination). One memory access is required when the Static Random Access Memory (SRAM) is either the source or destination.

2. *CPU*: Each transfer requires an explicit **mov** instruction, which spans three 16-bit words in the MSP430 instruction set. Therefore, each transfer takes five clock cycles (three cycles to fetch the **mov** instruction and two cycles for the actual transfer).
3. *ISR*: Same as in CPU but requires additional overhead of entering and exiting the interrupt. At a minimum, this includes saving and restoring the current system state to the software stack. Details of interrupt handling are provided in [16].

Table 3.1: Number of clock cycles needed to transfer one word between main memory and accelerator.

Method	Number of clock cycles to transfer one word	Number of main memory accesses to transfer one word
1. DMA	2	1
2. CPU	5	4
3. ISR	5, plus 6 cycles initialize ISR	5, plus 5 accesses to enter/exit ISR

Table 3.1 and the measured energy per cycle of the CPU, DMA, and memory allows us to calculate the energy spent on data transfers. This gives insight into how much energy is consumed by the accelerator doing the actual computations compared to the data transfer energy. Figure 3-1 and Figure 3-2 plot the energy breakdown for the four accelerators based on chip measurements. The following explains the five categories in the two figures.

a) *Accelerator*: Energy consumed by the accelerator only, not including bus or data transfer energy.

b) *Memory*: Energy consumed by the main memory. In methods 2 and 3 (CPU and ISR), this includes the energy of fetching the mov instruction from memory.

c) *Transfer*: Energy consumed by the CPU or DMA logic, computed as the energy per cycle $\times$ number of clock cycles (Table 3.1).

d) *ISR Entry*: Energy consumed by the CPU when entering an ISR.

e) *ISR Exit*: Energy consumed by the CPU when exiting an ISR.

The energy breakdown serves to illustrate some trade-offs between the three methods. In all cases the DMA is the most energy-efficient method for data transfers. However, typically a processor contains a limited number of DMA channels (e.g. the biomedical processor provides 3 channels). More channels may be required to support applications in which many modules work simultaneously, at the expense of higher DMA power. Further,

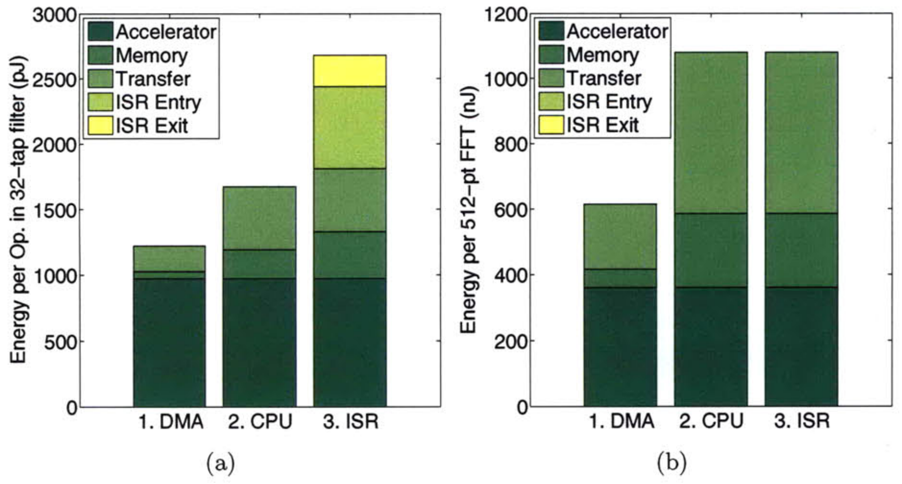


Figure 3-1: Breakdown between the computation energy in the accelerator and the energy expended in transferring data to/from (a) FIR and (b) FFT. Data transfers are performed with DMA, CPU, and interrupt service routine.

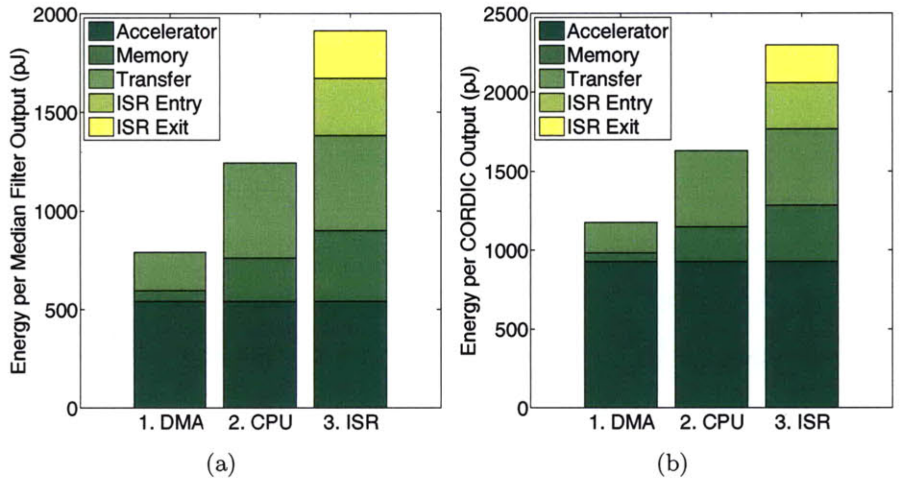


Figure 3-2: Breakdown between the computation energy in the accelerator and the energy expended in transferring data to/from (a) Median and (b) CORDIC. Data transfers are performed with DMA, CPU, and interrupt service routine.

when the DMA has use of the shared address/data bus, the CPU stalls since it cannot fetch the next instructions from the SRAM. Consequently, in a system with many DMA channels, bus arbitration between the DMA and CPU must be carefully managed.

From a software point of view, the ISR method offers the most convenience and flexibility because the programmer can define a set of tasks to be done automatically (in the interrupt routine) whenever the accelerator outputs are available. However, from an energy point of view, ISRs are less efficient than the other two methods since entering and exiting interrupts can impose sizeable energy costs. The interrupt handling overhead is especially significant for accelerators which consume/provide one word at a time (e.g. FIR and median filters). Conversely, the overhead is negligible in block-based processing (e.g. FFT) .

To summarize, Figure 3-1 and Figure 3-2 show that the data transfer energy can exceed the computation energy, particularly when the CPU is employed. This indicates that the accelerators are efficient relative to the rest of the system, and sizable gains can be achieved by decreasing the data transfer energy. One component of the data transfer energy involves driving the large address and data buses connecting all peripherals, which will be discussed in the next section.

3.2 Bus Architecture

In microcontroller architectures, peripherals such as timers and I/O ports are typically linked to the CPU via a shared bus, for example in the 8051 [67], MSP430 [68], and PIC [69]. A higher-complexity microcontroller, the 32-bit ARM Cortex-M3 [70], utilizes dedicated buses between the CPU and the program and data memory, but all the peripherals remain connected by a peripheral bus to the CPU. In this work, accelerators introduce additional capacitive loading on the shared bus. More generally, this is an important consideration as more specialized modules are being added to modern systems such as multimedia processors and microcontrollers. Accordingly, this section analyzes the energy impact of placing some modules on a separate bus.

It should be noted that the implemented biomedical processor utilizes a single-bus structure. However, here we make use of the processor design and layout to extract parameters for a bus energy model. This model is in turn used to examine the impact of an alternate bus structure.

Figure 3-3 shows the basic single-bus structure. All peripherals are memory-mapped, implying that the CPU can access the peripherals in the same manner as accessing memory. In the following discussion, we will refer to the set of address and data signals connecting more than one module as one bus. A set of address lines, which only the CPU and DMA can control, connects the CPU and all other modules. In this section we consider uni-directional data lines: the CPU broadcasts data to all modules on one set of data lines, and each module provides output on its own set of data lines which are then multiplexed before being read by the CPU. However, the analysis below can be easily modified for tri-stated, bidirectional data lines.

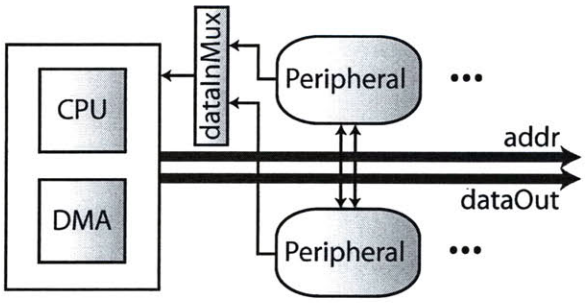


Figure 3-3: Basic bus structure with all peripherals connected to the same address and data signals driven by either the CPU or DMA. Each module outputs data on its own data lines, which are multiplexed into the CPU.

An alternate scheme is shown in Figure 3-4, in which the accelerators are removed from the primary bus and put on a secondary bus. A second DMA is added to the secondary bus to allow data transfers directly between two accelerators without CPU intervention. In general, this structure has the advantage of reduced capacitive loading on both primary and secondary buses, so the energy to switch either one of these buses is smaller compared to the original structure. However, extra energy in interfacing two buses is required when two modules on different buses need to communicate. The average switching energy of this bus structure, therefore, depends on the software application.

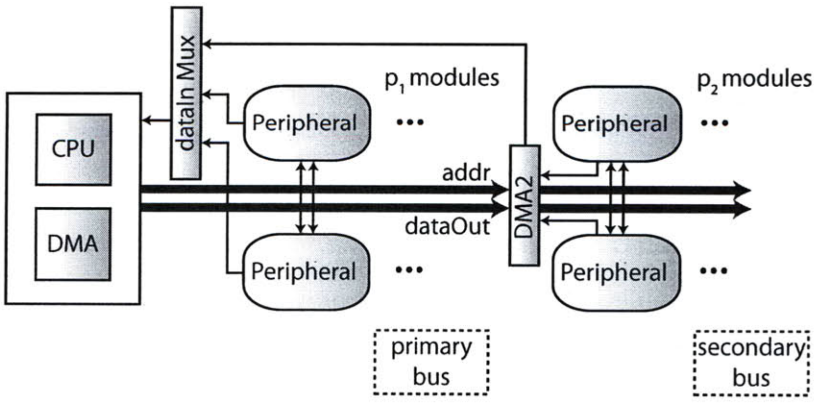


Figure 3-4: Alternate bus structure with accelerators connected by secondary bus and all other blocks on the primary bus. The two buses are interfaced by a bridge.

The average switched capacitance per cycle of the two bus structures can be modeled as:

$$C_{1bus} = \alpha(C_{CPU} + C_{peri}) \tag{3.1}$$

$$C_{2bus} = \alpha\left[(1-y)(C_{CPU} + p_1 C_{peri}) + (1-x)(p_2 C_{peri} + C_{DMA2}) + (1-x-y)C_{interface}\right] \tag{3.2}$$

where

- α: the average bit switching activity on the bus (i.e. the average number of bits that transition from 0 to 1 in a clock cycle divided by the total number of bits).
- C_{CPU}, C_{peri}: the capacitive load on the bus from the CPU and a peripheral respectively.
- x and y: the percentage of clock cycles where only the main or secondary buses are active respectively.
- C_{DMA2}: the switched capacitance in the secondary bus controller (DMA2).
- $C_{interface}$: overhead of transferring data between the two buses.

We determined the capacitance parameters from the layout of the biomedical processor including interconnect parasitics. C_{DMA2} is taken from the switched capacitance of the DMA implemented in the processor. In the platform, the number of modules on the primary bus (p_1) is 9, and the number on the secondary bus (p_2) is 4. α is averaged over all the bus transactions of the processor while it executes a benchmark application. The power of multiplexing the data lines (dataInMux in Figure 3-4) is similar in both structures.

Substituting these values into the above model, we obtain the plot in Figure 3-5 for $C_{1bus} - C_{2bus}$ as a function of x and y, indicating the switched capacitance savings under the alternate two-bus structure. Since x and y differ between applications, estimated values for several biomedical applications – heart sound [40], EEG feature extraction [19], and pulse-oximetry [39] – are shown on the plot. An average bit switching activity (α) obtained from chip measurements is used for all three applications. $C_{1bus} - C_{2bus}$ for these three applications are -2.7pF, -0.39pF, and 0.47pF respectively.

In applications where most transactions occur only on one bus, the alternate dual-bus structure provides energy savings as expected. However, in applications that require transactions between the two buses, the overhead of interfacing the two buses causes an overall energy increase.

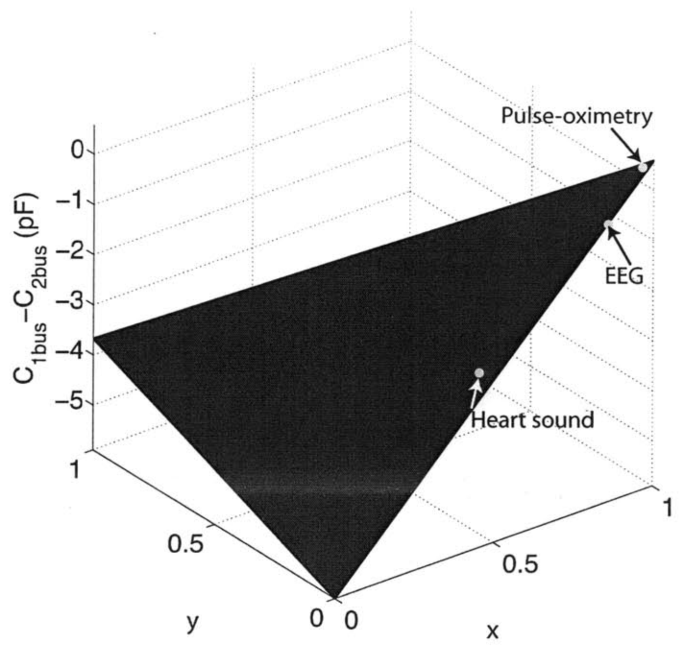


Figure 3-5: Difference in average switched capacitance between 1) using one bus to connect all peripherals and 2) putting accelerators on second bus. x and y respectively denote the percentage of clock cycles where only the main or secondary buses are active in a given application. Values for several applications are shown.

We now generalize this analysis to the case where any module can be assigned to any one of two buses. Depending on the typical bus usage characteristics under consideration, certain configurations will be more energy-efficient than others due to a smaller average switched capacitance. The average switched capacitance in a given bus configuration can be modeled by Equation 3.3.

$$
\begin{aligned}
C_{bus-avg} = \frac{1}{S}\sum_{i=1}^{N}\sum_{j=i+1}^{N}\alpha_{ij}T_{ij}\Bigg[\quad & (1-x_i)(1-x_j)\sum_{k=1}^{N}(1-x_k)C_k + \\
& x_i x_j\left(C_{DMA2-init} + \sum_{k=1}^{N} x_k C_k\right) + \\
& (x_i + x_j - 2x_i x_j)\left(\sum_{k=1}^{N} C_k + C_{interface}\right)\Bigg] \qquad (3.3)
\end{aligned}
$$

where

- x_i: set to 0 if module i is assigned to the primary bus (bus 1), and set to 1 if assigned to the secondary bus (bus 2)

- C_i: the load capacitance that module i adds to the bus
- $C_{DMA2-init}$: overhead of initializing DMA2 (Figure 3-4) to do transfers within bus 2.
- α_{ij}: the bit switching activity when modules i and j communicate
- T_{ij}: the number of bus transactions between modules i and j per unit time in a given application.
- S: total number of bus transactions per unit time.

Again, the capacitance parameters are extracted from the biomedical processor layout. For the numerical results below, we set α_{ij} to the average bit switching activity from the address/data traces of the processor as it runs an EEG feature extraction application [23]. However, the model allows a different α_{ij} for each pair of modules.

In Equation 3.3, the first term in the double summation gives the switched capacitance during one transaction between modules i and j if they are both assigned to bus 1. Similarly, the second term handles the cases when both modules are assigned to bus 2. The third term accounts for the capacitance when the modules are on different buses, which includes the capacitance on both buses and the overhead of interfacing them. It is assumed that the energy of the two bus controllers are similar and has weak dependence on the number of modules on the bus. Hence the energy difference between bus configurations arises only from discrepancies in the load capacitance and the interface overhead.

Having the above model for one assignment of $x_i, i \in 1..N$, we can now evaluate it over different bus configurations to find the one with the lowest switched capacitance, and thus the lowest switching energy. In general this is a binary optimization problem with a cubic cost function which is difficult to solve algorithmically. Fortunately, because of the low dimensionality of this problem (there are 13 modules to be assigned), an exhaustive search can be computed quickly. Note that the problem is symmetric and therefore one module can be arbitrarily assigned to one bus to halve the search space. We assign the CPU to bus 1 and then enumerate the 2^{12} possibilities for the remaining 12 modules. Due to the relative simplicity of Equation 3.3, the search can be completed within several seconds.

The model is evaluated using bus transaction data from an EEG feature extraction application (Figure 6-7) that will be detailed in Section 6.3. The bus configuration that minimizes the bus energy involves assigning the CPU, SRAM, FIR, and general-purpose I/O ports to bus 1 and the remaining modules to bus 2. Figure 3-6 plots the histogram of average switched capacitance across all possible bus configurations (2^{12} in total) for the EEG feature extraction application. Towards the right of the plot, the large bin contains configurations where the memory is assigned to a different bus than the CPU. The average switched capacitance in a single-bus structure is 2.42pF. The minimum average switched capacitance, in the configuration described above, is 0.754pF, a 3.2-fold decrease. Although this turns out to not be a large percentage of the total switched capacitance in the biomedical processor, the analysis above can be applied to other processors with significant bus energy. In addition, the interconnect capacitance in a large shared bus becomes increasingly important in more advanced process technologies.

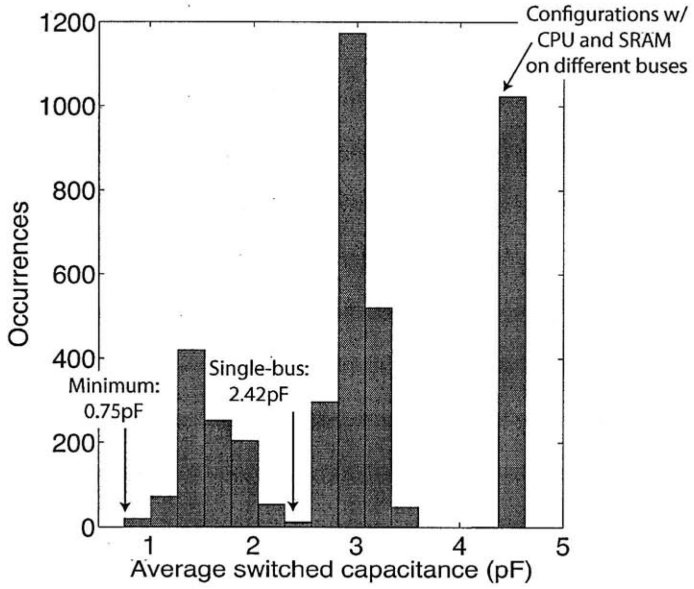


Figure 3-6: Histogram of average switched capacitance across 2^{12} bus configurations for EEG application. The bar on the right contains configurations where CPU and SRAM are assigned to different buses.

Here, the model was applied to one algorithm as an illustration, but multiple applications can be easily supported by adding the bus transactions (T_{ij}) over different applications before evaluating Equation 3.3. Finally, it is important to remember that the interfacing between the two buses introduces additional delay. If this delay lies in the critical path and the resulting slow down is unacceptable, the logic synthesis tool will increase logic cell sizes in the critical path, which reduces the energy savings. Thus the two-bus partition is advantageous when the critical path does not involve the interface between the two buses.

3.3 Power Gating

Although peripherals and accelerators can perform specific tasks more efficiently than the general-purpose CPU, thus providing active energy savings, the additional blocks impose leakage overhead when not in use. One effective method to reduce leakage overhead is to power off the modules during idle periods, in a well-known technique known as power gating. This is accomplished by adding low-leakage power switches between the logic and the actual power source, as illustrated in Figure 3-7. A group of logic cells sharing a virtual supply is said to belong to the same power domain.

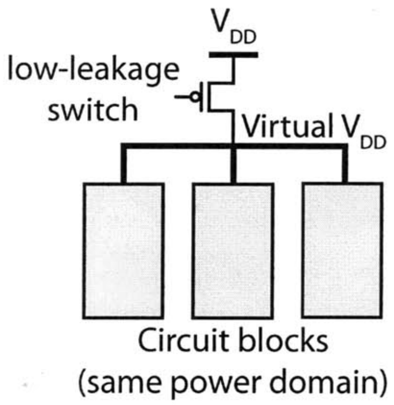


Figure 3-7: Illustration of power gating.

3.3.1 Power Gating Implementation

Before discussing power gating trade-offs in this design, we briefly describe how power gating was implemented on this platform. Each module shown connected to a power switch in Figure 1-4 can be powered on/off independently. The power switch is physically realized as several $50\mu m$ wide switches connected in parallel and distributed along the virtual V_{DD} supply of the module. The SRAM has two low-leakage modes summarized in Table 3.2. The CPU is not powered gated since doing so would cause internal CPU states to be lost, and there is no provision in this platform to save the internal CPU states to a separate location before powering off.

The Power Management Unit (PMU) manages the power on/off sequence of all modules on the chip. The sequence of signals is shown in Figure 3-8, along with the finite state machine in the PMU which generates the signals. The power off sequence involves:

Table 3.2: Two modes to reduce leakage in the SRAM.

SRAM Component	Retention Mode	Power-Gated Mode
Global timing control	off	off
Address decoders	off	off
Column periphery	off	off
Row periphery	on	off
Bit-cell array	on	off

- enabling isolation cells to force outputs of the module to "0" or "1"
- turning off power switch

The power on sequence involves:

- turning on power switch
- asserting module's reset signal
- turning off isolation cells and de-asserting reset signal

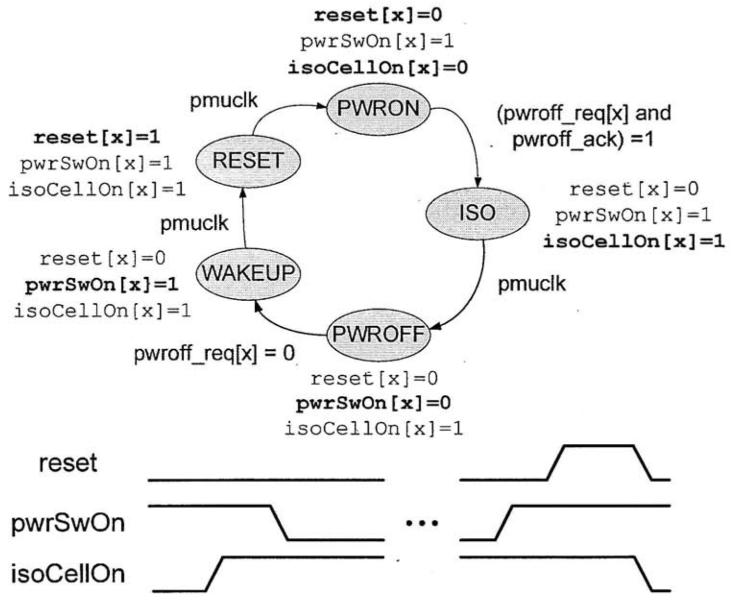


Figure 3-8: Power on/off sequence managed by the Power Management Unit.

Isolation cells serve to drive all outputs of a power-gated module to logic "0" or "1" levels. Otherwise, excessive leakage will occur if the outputs float to an intermediate value while driving other circuits that are still powered. The isolation cells in our platform are simply AND or OR cells with one input tied to the isolation signal managed by the PMU.

The application software can control power gating by two means. First, it can directly write to registers in the PMU to power a specific module on or off. Additionally, power switches can be configured to turn off/on automatically when the platform enters/exits a system-level low power mode.

3.3.2 Break-even Time

As one might expect, power gating imposes an energy overhead that must be considered before making a decision to power off a module. This overhead arises from the energy needed to turn power switches on and off, the energy for the power management logic, the need to restore charge back to a domain after powering it back on, as well as the cost of re-initialization. Due to this overhead, once powered off, a power domain should be kept in the off state for a minimum period of time known as the break-even time in order to achieve energy savings from power gating. More formally, the break-even time is given by

$$P_{leak}T_{be} = E_{switch} + E_{PM} + E_{restore} + P_{sleep}T_{be} \tag{3.4}$$

$$T_{be} = \frac{E_{switch} + E_{PM} + E_{restore}}{P_{leak} - P_{sleep}} \tag{3.5}$$

where P_{leak} and P_{sleep} refer to the leakage power of a power domain when its power switch is on and off respectively. E_{switch}, E_{PM} and $E_{restore}$ represent different components of the power gating energy overhead as described above.

A thorough account of the energy transfers that occur during power gating is provided in [42], which noted that E_{switch} and $E_{restore}$ are not constants but instead depend on how long a domain has been powered off. Since our platform employs on-chip power switches, characterizing E_{switch} and $E_{restore}$ for a domain is very difficult. Fortunately, the break-even time can be measured experimentally without knowing the exact values of these parameters using the method proposed by [42] and outlined as follows. In our platform, although the power switches are not accessible externally, the CPU can execute a short assembly program to toggle the power switch of a domain periodically. Then, we run a second program with the same instructions, except the power switch is kept on rather than toggled. We vary the clock frequency until the average power of these two programs are equal. Specifically,

$$P_{program1} = P_{cpu}(t_{on} + t_{off}) + P_{domain}t_{on} + E_{overhead} + P_{sleep}t_{off} \tag{3.6}$$

$$P_{program2} = (P_{cpu} + P_{domain})(t_{on} + t_{off}) \tag{3.7}$$

where t_{on} and t_{off} are the times over which the power domain is turned on and off in program 1. P_{domain} and P_{sleep} denote the domain leakage power when the power switch is on and off respectively. Since $P_{program1} = P_{program2}$, the terms involving t_{on} cancel and we have

$$P_{domain}t_{off} = E_{overhead} + P_{sleep}t_{off} \tag{3.8}$$

The t_{off} which equates the power of the two programs is the break-even time. Using this method, we measure the break-even times for the accelerators as listed in Table 3.3. It is shown that power gating the accelerators is advantageous if the off state persists for at least several hundred μs. Since physiological signals typically have data rates of less than 1kHz, these relatively low break-even times imply that the accelerators can be powered off after processing each incoming sample or after a block of several buffered samples.

Table 3.3: Break-even time of accelerators measured at 1V.

Module	Break-Even Time
FIR	$427\mu s$
FFT	$1.02ms$
CORDIC	$512\mu s$
Median	$461\mu s$

3.3.3 Power Gating Granularity

In this section, we examine the trade-off between coarse- and fine-grained power gating. In a coarse-grained approach, several modules are grouped into one power domain and are power-gated together. In a fine-grained approach, each module forms its own power domain and can be independently powered off. Power gating on an even smaller scale – for groups of logic gates within a module – has been investigated in [71] and [72]. However, sub-module gating is much better suited to circuits with a regular structure (an FPGA and memory address decoder in [71] and [72] respectively) than the generic logic blocks in this platform. Authors of [72] proposed a methodology to synthesize arbitrary logic with automatic power gating; however, the area and power overhead of the control mechanism can be significant.

Table 3.4 summarizes a qualitative comparison of coarse- and fine-grained power gating, with numerical results to follow. The power management unit described previously requires one Finite State Machine (FSM) for each power domain, hence the control cost scales linearly with the number of domains. However, isolation cells must be inserted at the output of each gated module regardless of the power gating approach. Likewise, the power switches are sized according to the modules' power consumption; by superposition, the total width of power switches would be the same whether coarse- or fine-grain power gating was used.

Table 3.4: Qualitative comparison of coarse- and fine-grained power gating.

Consideration	**Comparison**
Energy in control logic	Scales approximately linearly with number of power domains
Number of isolation cells	Same for both approaches.
Total width of power switches	Same for both approaches.
Leakage reduction factor	Same for both approaches.
Total leakage energy savings	Higher for fine-grained power gating

From an energy point of view, fine-grained power gating incurs higher control energy but provides more opportunities to power off a module to save leakage energy. Therefore, if a module can be powered off for a sufficiently long period of time as a separate power domain than as part of a larger domain, then the fine-grained approach is beneficial. We denote this additional power gating opportunity in a fine-grained approach as T_{fg}. From simulation of the PMU, we find that the control energy per power gating cycle is less than 1pJ at 1V. Accordingly, Table 3.5 lists T_{fg} for different modules in the platform. For large blocks such as the median filter and FFT, the control cost can be recouped after several clock cycles of power gating, and therefore these modules should be placed in their own power domains. Heavily used modules likely to be powered for more than several hundred clock cycles, such as FIR, GPIO, serial port, and timer, should also be placed in separate domains so that they do not prevent other blocks from being shut off. This leaves four modules which can conceivably be grouped into one power domain, but for additional flexibility, they are kept in separate domains.

3.3.4 Power Switch Selection

One important consideration in power switch selection is the choice between a PMOS header and an NMOS footer. The use of header and footer switches are illustrated in Figure 3-9.

Table 3.5: T_{fg} for modules in the processing platform.

Module	T_{fg} (μs)	T_{fg} (clock cycles at 1V)
Median	0.44	4.4
CORDIC	5.74	57.4
FFT	0.356	3.6
FIR	4.13	41.3
GPIO	7.92	79.2
Serial port	6.59	65.9
Real time clock	6.84	68.4
Timer	12.1	121
S/W debug support	19.6	196
Watchdog timer	18.6	186

In the context of a voltage-scalable design, a header switch has the following benefits and shortcomings relative to a footer switch.

Benefits

- Does not require special standard cell layout; compatible with typical cell layouts.
- Can provide lower leakage at low V_{DD} (0.5V) if the gate of the power switch is driven by nominal or I/O V_{DD} (1V or 1.8V) such that its V_{GS} is negative (see Figure 3-9).
- Can apply forward bias when switch is *on* to improve on-current and reverse bias when switch is *off* to reduce sub-threshold leakage current. This is not possible for an NMOS in an N-well process.

Shortcomings

- For the same switch size, a PMOS header has lower on-current than NMOS, resulting in a larger speed penalty in the power-gated circuits.
- For the same speed penalty, a header must be wider, increasing the energy overhead of turning the switch on and off.
- At 0.5V, leads to significantly slower speed than a footer, since a footer can be driven by nominal or I/O V_{DD} such that its $V_{GS} > V_{DD}$ (see Figure 3-9).

The delay and leakage current (I_{off}) trade-offs in a 0.13 μm process are quantified in Table 3.6. We simulate a 32-bit adder without power gating, with a header, and with a footer. Both header and footer are sized so that the voltage drop across them are the same, and the adder delay increases by 5% relative to the non-gated adder at the nominal supply

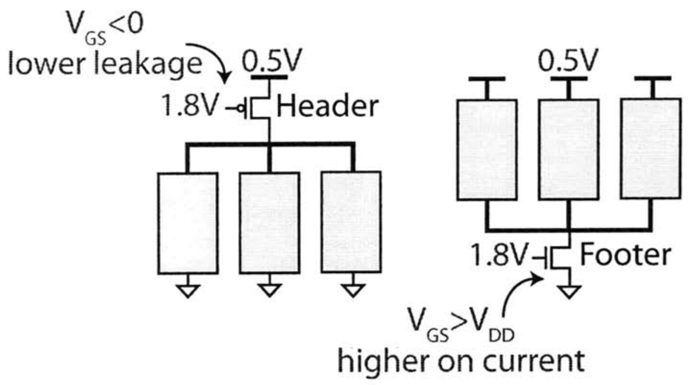


Figure 3-9: Illustration of using PMOS header or NMOS footer as power switches.

voltage (V_{DD-HI}). The delay and I_{off} at a low voltage (V_{DD-LO}) are simulated with the header and footer both driven by a $0 \rightarrow V_{DD-HI}$ signal.

Table 3.6: Comparison of PMOS header and NMOS footer switches in a 0.13 μm process.

	No power gating	PMOS header	NMOS header
Normalized switch size	N/A	2.58	1
V_{DD-HI} delay	1	1.05	1.05
V_{DD-HI} I_{off}	1	1/107	1/485
V_{DD-LO} delay	1	1.16	1
V_{DD-LO} I_{off}	1	1/10200	1/507

Like the selection between header and footer, switch sizing embodies a trade-off between speed during active mode and leakage current during idle mode. When a module is active (i.e. power switch is *on*), the voltage drop across the switch should be small, which favors a wide power switch with smaller equivalent *on* resistance. When a module is power-gated (i.e. power switch is *off*), the switch should be small to limit leakage currents. This platform employs a common approach to switch sizing – the switches are sized sufficiently large to achieve a fixed IR drop across the switch, and thus a fixed speed penalty compared to a design without power gating. In this platform that operates between 0.5V and 1V, the switches are sized for a 15% speed penalty at 0.5V. Because the switches are high-V_t devices that are close to being in sub-threshold at 0.5V, designing for a speed penalty below 15% at 0.5V would lead to very large power switches and corresponding increases in leakage power. To size the switches, we first determine the voltage drop corresponding to 15%

speed penalty at 0.5V, then find the required switch width from a simple model:

$$\Delta V = I_{avg} R_{eq-unit} N,$$

where I_{avg} is the average current drawn by the power domain, $R_{eq-unit}$ is the *on* resistance of a unit-sized switch ($W = 50\mu m$), and N is the number of parallel unit-sized switches required by the power domain. Note that the power switches are sized to provide the *average* current drawn by a power domain; the switches need to be substantially larger if they need to supply the *peak* current. To support peak current requirements during circuit switching, each domain also contains local decoupling capacitors between the virtual supply and ground, which are sized to provide the necessary charge. The presence of decoupling capacitors increases the break-even time of a power domain, and this was included in the break-even time measurements in Section 3.3.2.

3.4 Conclusions

This chapter examines the overhead of including accelerators into a typical low power processor architecture. Although the discussion is motivated by accelerators, the analysis and conclusions are more broadly applicable to any functional module one wishes to include in the platform.

Similar to other low power processors, the biomedical processor offers three methods to transfer data to and from the accelerators (as well as to other modules). Trade-offs between energy, flexibility, and convenience exist between these methods. An energy breakdown based on chip measurements shows that a DMA is the most energy-efficient way to transfer data to and from the accelerators. An interrupt service routine (ISR) is the least efficient because of the overhead to save and restore the CPU state. However, an ISR offers the convenience of allowing the programmer to define a set of tasks to be performed automatically each time an accelerator output becomes available. We find that in the latter approach, the energy to transfer data can exceed the total energy spent on computation by the accelerator. This has two implications for system design. First, more DMA channels should be included (in addition to the three channels in the biomedical processor). However, bus arbitration between the CPU and DMA should be carefully managed. Second, optimizing data transfers will have a large impact on the overall system energy.

Accelerators introduce additional capacitive loading on the shared bus. When all modules are connected to the same bus, communication between any two modules switch the load capacitance of other modules. To circumvent this, we consider an alternate dual-bus structure where the accelerators are placed on a separate bus from the CPU and other peripherals. Next, we generalize this to the problem of assigning every module to any one of two buses. We present a framework for analyzing the bus energy, considering that applications have distinct bus transaction characteristics. Applying this to an EEG application, the minimum-energy bus partitioning decreased the bus energy by approximately 3$\times$ over the classic single-bus structure. Although this was a small percentage of the total energy in the biomedical processor, the same analysis can be used for other processors where the bus energy is more prominent.

While power gating is a well-known technique to reduce idle leakage power, this implementation is uncommon amongst low power microcontrollers in that each module can be individually power gated with on-chip switches. Accordingly, we consider the trade-off between module-level power gating as implemented, or a coarser-grained approach. It is shown that the control overhead of module-level gating is low and can be compensated by the leakage energy savings after less than 200 clock cycles. Therefore, most modules can benefit from being independently powered off.

Chapter 4

Voltage-Scalable Logic Design

The previous chapters have discussed how to reduce energy in different components of the biomedical processor. A major opportunity to improve the energy efficiency over the entire processor, and in digital circuits in general, is to aggressively scale its supply voltage according to performance demands. When the supply voltage is scaled to below the transistor threshold voltage (V_t), the transistor is said to enter the sub-threshold or weak inversion region of operation.

The idea of exploiting weak inversion for low power circuits was pioneered by Vittoz in the 1960's [73], with one of the earliest applications being electronic wristwatches [74]. Sub-threshold operation has captured interest in recent years as a means to achieve quadratic energy reduction in the active (switching) energy. Although circuits exhibit slower speeds at low supply voltages, the trade-off remains attractive for energy-constrained systems with relaxed throughput constraints. When V_{DD} approaches the sub-threshold region, the leakage energy per operation increases as a result of the leakage power being integrated over exponentially longer clock periods, as shown in Figure 4-1(a) for an arithmetic logic unit (ALU) in 65nm. These opposing trends in active and leakage energy give rise to a minimum energy point, or the optimal V_{DD} which minimizes the energy per operation [75]. Expressions for the optimal V_{DD} and V_t were given in [76], which also noted that the minimum energy point depends on the relative proportions of active and leakage energy – a high proportion of active energy (high activity factor) tends to decrease the minimum energy point. For many practical circuits, this point lies in the sub-threshold region.

The previous argument assumes that a circuit can complete a task at the speed achievable at the minimum energy point, and then shut off, so that it consumes negligible energy during idle periods. However, certain system components, such as memories, must be powered for arbitrarily long periods unrelated to their own speeds. In this case, it is essential to minimize the leakage power. Voltage scaling is also beneficial here, as it causes a decrease in leakage current by alleviating Drain-Induced Barrier Lowering (DIBL), which, combined with V_{DD} reduction, can provide an order of magnitude leakage power savings (Figure 4-1(b)).

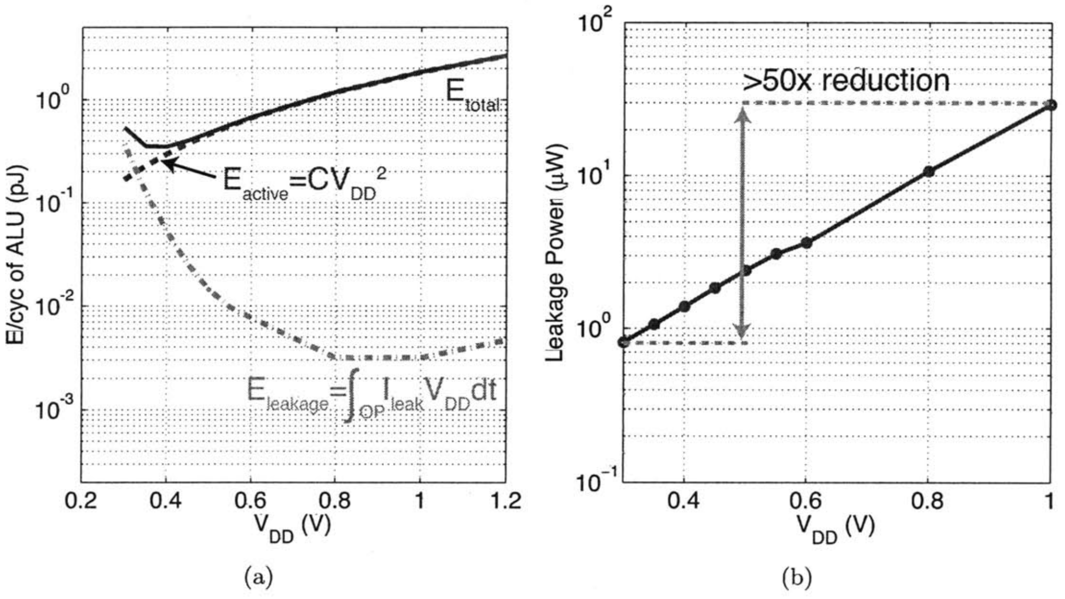


Figure 4-1: (a) Energy per clock cycle versus V_{DD} of an ALU in 65nm. In this circuit, there is an optimal V_{DD} which minimizes the energy per cycle. (b) Leakage versus V_{DD} in a 65nm SRAM.

As one might expect, voltage scaling presents great opportunities but also significant and interesting challenges in logic design. This chapter describes issues related to designing logic circuits at low voltages, starting with the impact of process technology on the minimum energy point. Subsequently, it addresses logic design and timing verification for two different types of low-voltage systems: 1) those that function mainly in the sub-threshold region (≈0.4V and below) and 2) voltage-scalable systems that operate from near-threshold up to nominal supply voltage. In the first type of system, we are primarily concerned with mitigating process variation which greatly impacts transistor behavior in sub-threshold. The design approaches described in this chapter were demonstrated in a 65nm microcontroller operating from 0.3V to 0.6V. The lessons from sub-threshold design inform our approach to voltage-scalable systems, one example being the biomedical signal processing platform. In particular, library design and timing considerations will be discussed. Lastly, the chapter presents measurement results of the 0.3V 65nm microcontroller.

4.1 Process Technology

In this section we explore two issues pertaining to process technology, the first being the variation in minimum energy point with technology scaling, and the second being the choice of technology based on the characteristics of the application.

4.1.1 Trend in Minimum Energy Point with Process Scaling

As noted previously, the minimum energy point depends on the relative contributions of active and leakage energy, and lies in sub-threshold for many practical circuits. Process scaling results in lower switched capacitance, but recent technology nodes have seen leakage current increasing substantially, in part because device thresholds are lowered to maintain performance while the nominal V_{DD} is scaled down. Therefore, a natural question is how the minimum energy point scales with process scaling, and whether it remains in the sub-threshold region.

To examine trends in the minimum energy point, we simulated a 32-bit Kogge-Stone adder [50] implemented with 65nm, 32nm, and 22nm predictive technology models. These models are adapted from the publicly available predictive models [77], courtesy of Professor Dimitri Antoniadis at MIT and Professor Yu Cao at the Arizona State University. The adder illustrated in Figure 4-2 was first designed and laid out in a 65nm process to extract wiring capacitance. In porting the adder to other technology nodes, the W/L of transistors are kept the same while L is scaled to the minimum length in the target technology. Wiring capacitance is scaled according to ITRS projections [78]. Since the switched capacitance and leakage current of a circuit are input-dependent, the average values over random input vectors are used in the discussion below.

The active energy per cycle versus V_{DD} of the adder is shown in Figure 4-3 for different technology nodes. As expected, the active energy decreases with scaling, but in advanced technologies wiring capacitance is a significant contributor to the total capacitance, and partially offsets the energy savings from lower gate capacitance. Changes in delay and leak-

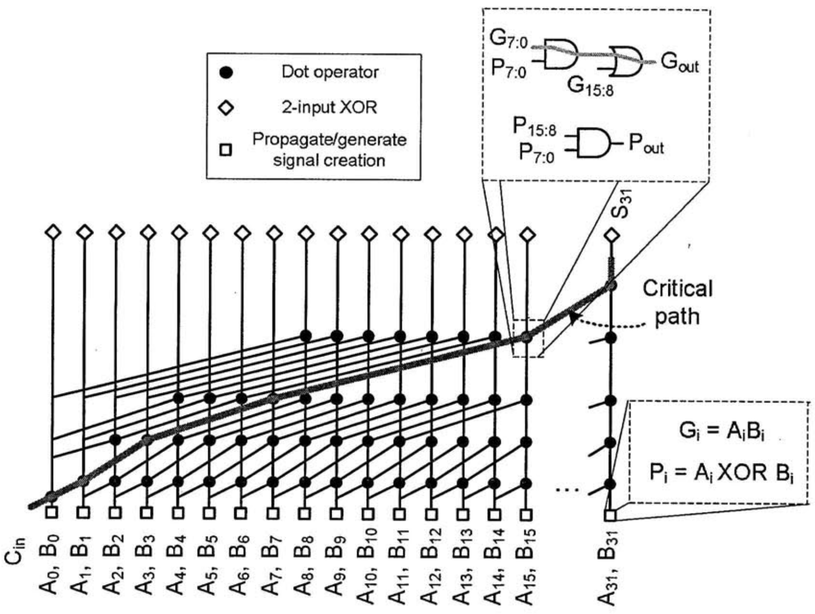


Figure 4-2: 32-bit Kogge Stone adder used in process scaling simulations.

age current with process scaling is shown in Figure 4-4(a) and Figure 4-4(b) respectively. The leakage energy per addition is equal to the leakage power integrated over the propagation delay of the adder, i.e., $\int_0^{t_p} I_{leak}V_{DD}dt$, and is plotted in Figure 4-5. Initially as V_{DD} decreases, the propagation delay increases linearly while the leakage power decreases super-linearly since both I_{leak} and V_{DD} are reduced. This causes a net decrease in the leakage energy per addition in the above-threshold region. However, in sub-threshold, the propagation delay increases exponentially at a faster rate than the leakage power is reduced, leading to a net increase in the leakage energy per addition.

Adding the active and leakage energy components gives the plot of total energy versus V_{DD} in Figure 4-6. The minimum energy point occurs at a higher voltage due to the larger proportion of leakage component at 32nm and 22nm. Nevertheless, the minimum energy point still occurs in the sub-threshold region, suggesting that aggressive voltage scaling will remain an important tool for reducing energy in future process technology nodes.

4.1.2 Technology Selection in a Given Application Scenario

The minimum energy point analysis above assumes that the circuit can operate at a reduced speed at the minimum energy point while still meeting application performance demands [76]. Further, since the leakage energy consists of the leakage power of the circuit integrated

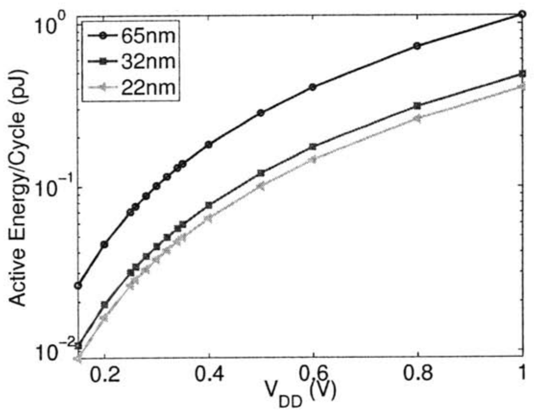
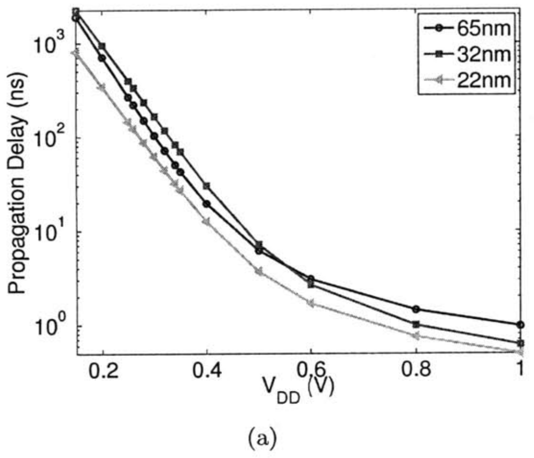


Figure 4-3: Active energy versus V_{DD} of a 32-bit adder simulated with predictive technology models.



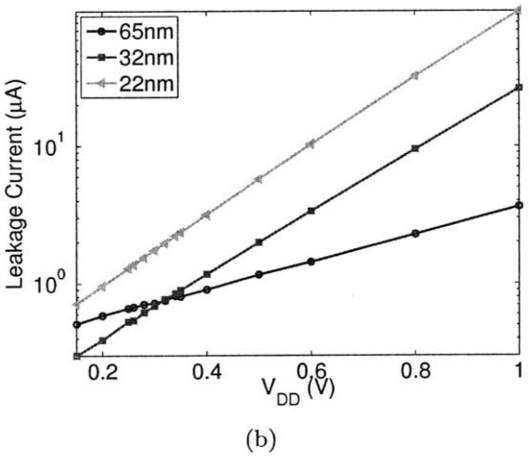


Figure 4-4: (a) Delay and (b) leakage current versus V_{DD} in a 32-bit adder simulated with predictive technology models.

over its propagation delay, it is implied that the adder is powered off immediately after use so that it consumes negligible leakage during idle periods. However, these assumptions are not always valid in practical usage scenarios – the circuit may need to operate at a higher speed in a given application. Here we generalize the analysis to consider application frequency constraints and duty cycle characteristics. The 32-bit adder and predictive models are again used for illustration, but this can be extended to other circuits and technologies.

In the following we use V_{DD} as the primary means to adjust the speed of a circuit. Although increasing device sizes in the critical path helps lower propagation delay, eventually the parasitic capacitance of the devices becomes significant and prevents further speed-up [50]. In the sub-threshold region, delay is an exponential function of V_{DD} and a linear func-

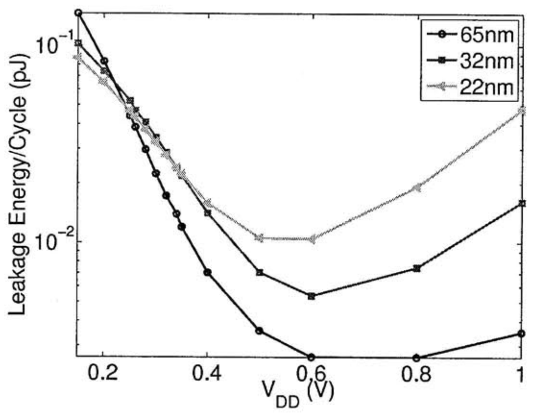


Figure 4-5: Leakage energy versus V_{DD} of a 32-bit adder simulated with predictive technology models.

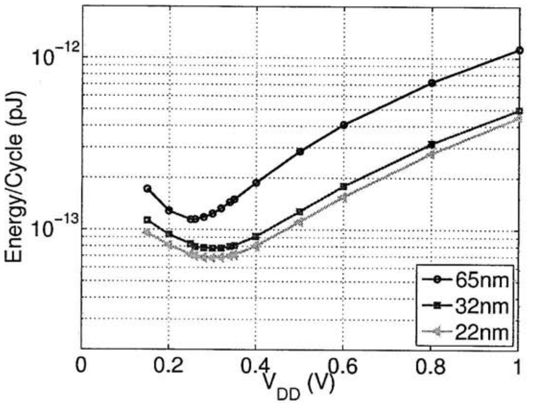


Figure 4-6: Total energy versus V_{DD} of a 32-bit adder. The minimum energy point occurs at a higher voltage in deeply scaled technologies due to an increased contribution of the leakage component to total energy.

tion of device width, and therefore raising V_{DD} is a more effective way to increase speed. Accordingly, the operating voltage of the adder is set by the application performance constraint, and this in turn determines the total energy per cycle. Figure 4-7 plots the energy required to meet the system clock frequency constraint in three technology nodes; each marker on the graph corresponds to a value of V_{DD}. At high frequencies, active energy dominates which favors smaller feature sizes. At low frequencies, leakage energy dominates, and the energy curves of the 22nm and 32nm adders begin to cross over. Note that if the clock frequency constraint lies below the minimum point of the curve (for example at 1MHz), the adder should be not be operated at this point. Rather, it should be operated at the minimum point of the curve and powered off afterwards.

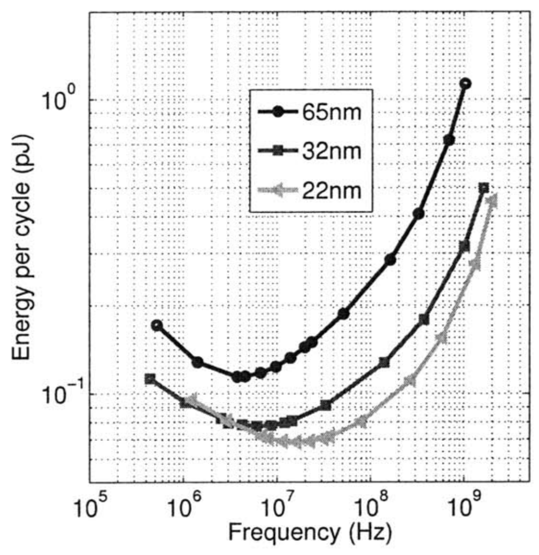


Figure 4-7: Total energy versus frequency of a 32-bit adder. Each marker corresponds to one value of V_{DD}.

Next we consider the impact of duty cycle. In an application with duty cycle d, the adder operates one out of $1/d$ clock cycles and is otherwise idle. If the adder can be powered off during its idle periods, the analysis is the same as before (Figure 4-7) since we only need to consider the leakage energy in one clock period. On the other hand, if the adder cannot be powered off (for example because it shares a supply with another component that is active), then we must consider the leakage energy over all $1/d$ clock cycles. Since this accentuates the leakage component of total energy, a process with lower leakage would be favored. Figure 4-8 plots the average energy per clock cycle of the adder with 10% and 1% duty cycle, where average energy is defined as $dE_{op} + (1-d) \int_{T_{cycle}} P_{leak} dt$. As the duty cycle decreases, the 22nm curve crosses the other curves at a higher frequency. Comparing the minimum achievable energy in all three processes, the relative advantage of 22nm lessens at low duty cycles.

The above analysis can be applied more broadly to larger systems and other process technologies. It should be noted that instead of simulating a large circuit at different technology nodes, an approximate analysis can be performed by abstracting the circuit into parameters including its average switched capacitance and leakage current versus V_{DD}, then extrapolating them to other technology nodes. The leakage current versus V_{DD} characteristic differs between process nodes since it depends on device properties like channel length

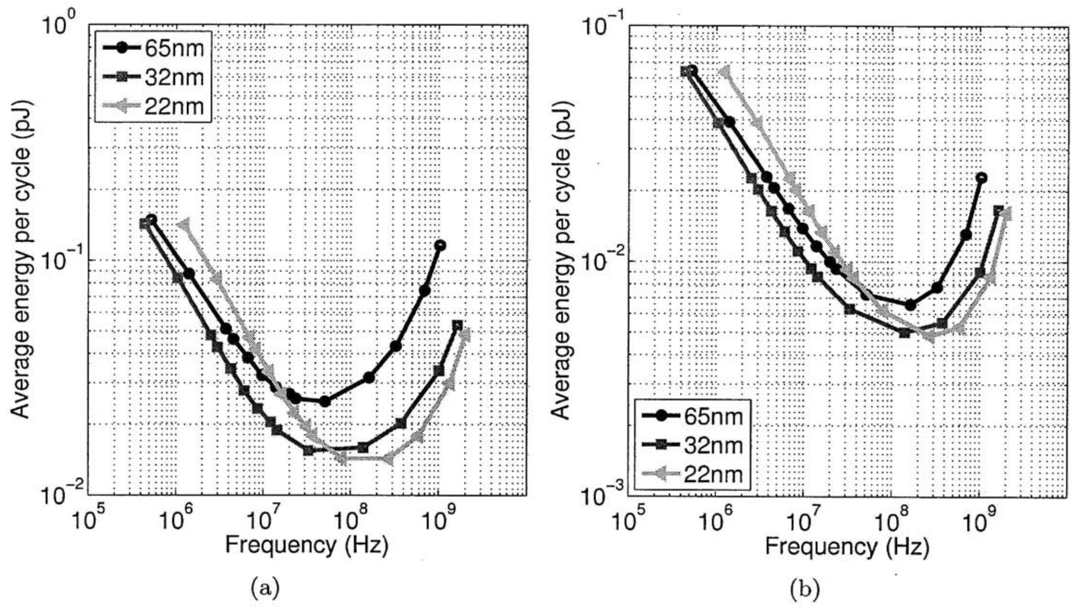


Figure 4-8: Average energy/cycle of adder at (a) 10% and (b) 1% duty cycle. Average energy is defined as $dE_{op} + (1-d)E_{leak/cycle}$.

modulation and DIBL, as well as circuit topologies like the number of stacked transistors. A estimated curve for each process node can be obtained by simulating the leakage versus V_{DD} of a simple circuit. Similarly, delay versus V_{DD} (needed to compute $E_{leak}/cycle$) can be obtained by simulating only the critical path of the circuit. A similar analysis was applied in selecting between high- and low-V_t options in the target $0.13\mu m$ process technology of the biomedical platform.

4.2 Standard Cell Library for Sub-threshold Systems

After determining the process technology, we now address the next building block of logic circuits, the standard cell library. The key challenge behind sub-threshold logic design stems from the exponential dependence of transistor current on V_t, and the variation in V_t found in modern process technologies. More precisely, a common model for the device current in sub-threshold is given by [79]:

$$I_D = I_o e^{\left(\frac{V_{GS}-V_t+\eta V_{DS}}{nV_{th}}\right)} \left(1 - e^{\left(\frac{-V_{DS}}{V_{th}}\right)}\right),$$

where

- n = the sub-threshold slope factor
- η = DIBL factor
- I_o = a process-dependent factor
- $V_{th} = kT/q$, the thermal voltage

Note that the drain current depends exponentially on both V_{GS} and V_t. In modern processes, V_t exhibits variation on both global and local scales. Global variation affects all transistors on a chip in the same manner; for instance, a chip at a strong-NMOS weak-PMOS global corner contains NMOS devices that are all stronger than average and PMOS devices that are weaker. Further, local variation implies that V_t of transistors on the same chip can also differ. In this section and the next, we will discuss the implications of variation on sub-threshold standard cell design and timing verification.

4.2.1 Logic Design Challenges

Sub-threshold logic design at a deeply scaled technology node must address two factors which critically impact functionality. First, random-dopant-fluctuation is a dominant source of local variation in sub-V_t, causing random, local threshold voltage shifts [80]. Moreover, the ratio of on to off currents in transistors (i.e. ratio of active to leakage currents) degrade by four orders of magnitude as V_{DD} is scaled from a nominal voltage to sub-threshold. This weak I_{ON}/I_{OFF} is exacerbated by exponential changes in device currents, and consequently static CMOS logic gates do not always provide rail-to-rail output swings. The two combined effects are illustrated in Figure 4-9 by the voltage transfer curve of a 65nm inverter at 300mV. Global variation weakens the NMOS devices relative to PMOS in this example, and skews the VTC towards one side. Additionally, local variation randomly changes the strengths of PMOS and NMOS to cause perturbations in the VTC, in some cases severely degrading the logic levels.

These degraded logic levels can adversely impact functionality, even in typically robust static CMOS circuits. For example, reduced logic swing in inverters I_2, I_3, I_4, I_5 of Figure 4-10 decreases the hold static noise margin (SNM) of latches in the classic transmission-gate register. Another failure mechanism is illustrated in the transient simulation of Figure 4-10. Here, because the clock buffer had reduced output swing, the transistor M_{11} cannot be completely turned off during the transparent mode of the slave latch. Consequently, a

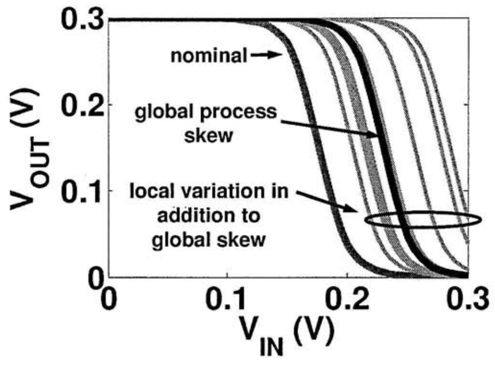


Figure 4-9: Effects of variation and reduced I_{ON}/I_{OFF} on sub-V_t inverter voltage transfer curve.

signal cannot propagate successfully from node N2 to N3.

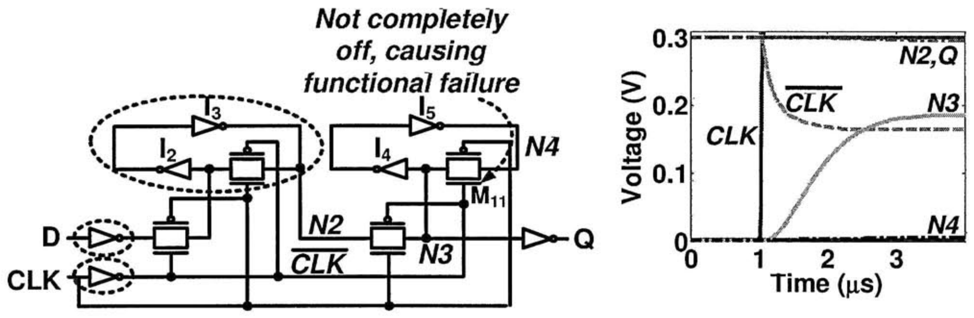


Figure 4-10: Reduced voltage swing in sub-V_t can impact hold SNM and signal propagation in registers. The latter issue is shown by transient simulation.

Previous work has presented different strategies and considerations in designing low-voltage standard cell libraries. In an early treatment of this topic, the authors of [81] sized logic gates for operation at the minimum possible V_{DD} based on the trade-off between strong-NMOS/weak-PMOS and weak-NMOS/strong-PMOS global process corners. It was also observed that certain cell topologies, where several parallel leaking devices fight one active device, exhibit degraded output voltages and are thus less suited to sub-threshold operation. In [82], authors found that the failure point of typical standard cells in $0.18\mu m$ is lower than the minimum energy point, and so minimum sized devices in standard cells should be optimal for minimizing energy. The work in [24] reached a similar conclusion for a $0.13\mu m$ process. There, authors compared processors implemented with libraries employing three device sizing strategies, and found that the same performance can be achieved with lower energy by increasing V_{DD} of the minimum-sized library rather than increasing transistor sizes.

The above work was performed on older process technologies in which local variation was less prominent. On the other hand, Figure 4-9 and Figure 4-10 have shown that local variation significantly degrades functionality at the 65nm node. One approach to mitigate local variation at the standard cell level is to increase the width and/or length of transistors, since empirical observations revealed that the standard deviation of V_t is proportional to the square root of the channel area ($\sigma(V_t) \propto 1/\sqrt{WL}$) [83, 84]. However, in the interest of minimizing energy, transistors should also be kept as small as possible, to lower CV^2 energy and leakage currents. A standard cell design approach to negotiate this trade-off is detailed in [85] and [86]. Since degraded output levels can cause standard cells and logic circuits to be non-functional, the approach determines whether a logic gate under design has adequate output levels by verifying it against the worst case gates in the library. The worst case gates are the ones that require its inputs to be closest to V_{DD} or ground in order to produce a correct output. For example, the worst case gates in a typical cell library would be the high fan-in NAND and NOR. As illustrated by their VTCs in Figure 4-11, the logic high input to NAND must be close to V_{DD}, and the logic low input to NOR must be close to ground, in order to turn on the NMOS- and PMOS-stacks in these cells sufficiently to produce the correct output. Under local variation, some cells will be able to drive the worst case gates, while some will have insufficient output logic levels. By performing Monte Carlo simulation with different transistor sizes in the gates under design, we can find the minimum sizes required for the gate to drive the worst case cells with high probability.

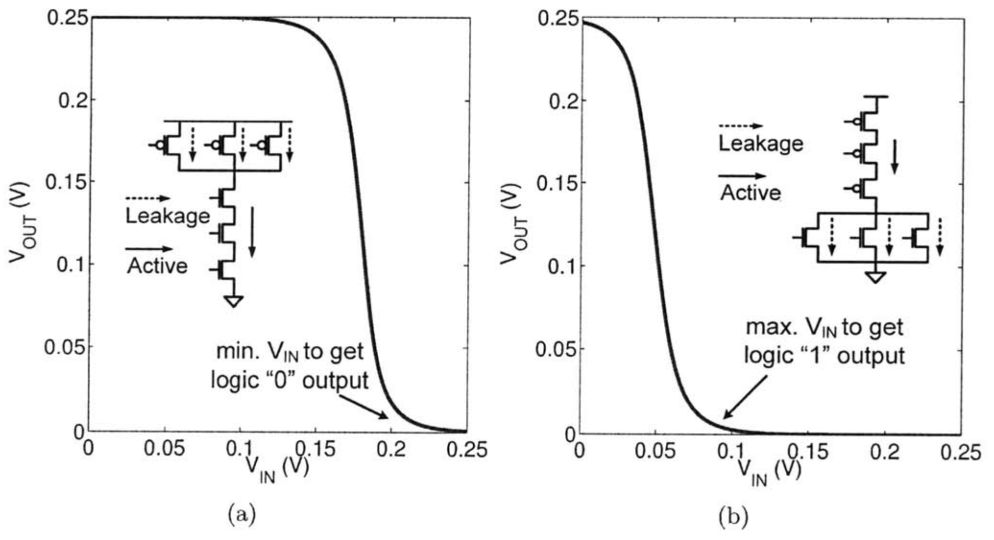


Figure 4-11: VTC of 3-input (a) NAND and (b) NOR cells.

4.2.2 Standard Cells for Sub-threshold Operation

A standard cell library functioning down to 0.3V in 65nm CMOS was developed in [86] and used to build the sub-threshold microcontroller test-chip that will be described in Section 4.6. Although the library was necessary to achieve functionality at 0.3V, its area and energy overhead was not addressed in [86]. Therefore, we will quantify the overhead in the following discussion.

Since the library design involved increasing the width of key transistors to mitigate local variation, we compare the area of the resulting sub-V_t cell library against a commercial low power cell library optimized for nominal V_{DD} operation. Figure 4-12 reports the ratio of total transistor area (WL) in the custom sub-V_t cells to that of the commercial cells. Since the cells with the lowest drive strength were re-sized in the sub-V_t library, while the higher strength cells are less susceptible to local variation, Figure 4-12 compares the total transistor area in the "$1\times$" strength cells of both libraries. Some sub-V_t cells such as XOR2 are larger than the commercial counterparts by several percent, while several cells are smaller. The 3-input NAND and NOR cells employ large devices for reasons explained previously. Not surprisingly, all sub-V_t flip-flop cells (*DFF*) occupy more area than above-V_t cells as well, since they represent an early point of failure in a library. Their back-to-back inverters (I_{2-5} in Figure 4-10) must be sized appropriately to ensure data retention, while the data and clock buffers must be sized such that they can properly turn the transmission gates on and off.

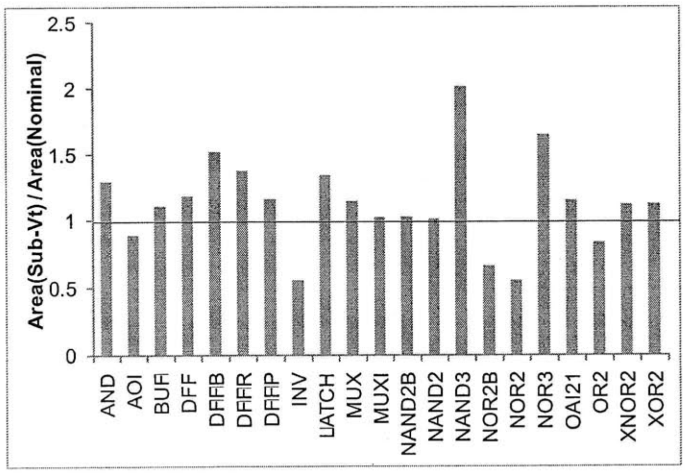
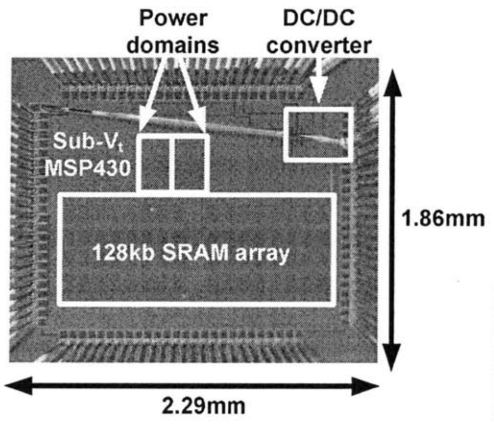


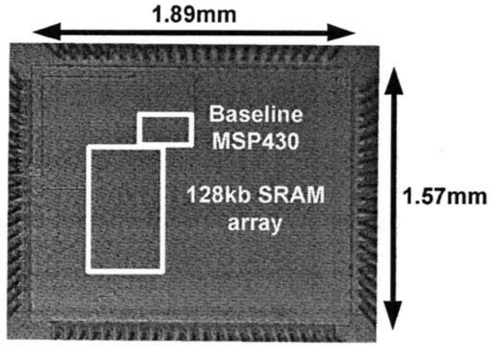
Figure 4-12: Ratio of total transistor area in the 65nm sub-V_t library cells to commercial library cells. Cells with $1\times$ drive strength are shown.

To evaluate the energy overhead, we implemented two versions of a 16-bit microcontroller core based on the MSP430 [68], one targeting low-voltage operation with the sub-V_t library (the sub-V_t chip) and the other targeting nominal-V_{DD} operation with the commercial library (the baseline chip). Both cores (approximately 10k gates in size) contain the same logic except for the memory interface since the two chips employed different SRAMs. The two cores were fabricated in a 65nm low power (LP) CMOS process, and their die micrographs are shown in Figure 4-13.

The chips were fabricated on two separate process runs with different global process characteristics[1] which prevented a fair comparison of the leakage power. However, we can characterize their switched capacitances to estimate the energy overhead of the sub-V_t library. In doing so, it is important that both designs are synthesized at speeds that allow comparison, since the synthesis tool tends to choose larger standard cells to meet aggressive frequency constraints. The baseline and sub-V_t chips are synthesized at the slow global corner with a clock period of 20ns at 1.1V and $200\mu s$ at 0.3V respectively. Since the inverter delay scales by approximately 9000× across these conditions, the chosen synthesis speeds allow a reasonable comparison of the switched capacitance. In Table 4.1, we summarize the measurement results. Note that the baseline chip does not operate below 0.8V while the sub-V_t chip is designed to function up to 0.6V, so we opted to compare measurements taken at the same frequency (1MHz). Results show that the use of the sub-V_t library in the 0.3V microcontroller leads to approximately 12% switched capacitance overhead.



(a)



(b)

Figure 4-13: Die micrographs of 16-bit microcontroller implemented with (a) sub-V_t library and (b) nominal-V_{DD} library.

[1]The sub-V_t chip was less leaky than average, and the baseline design was the opposite.

Table 4.1: Comparing the switched capacitance of the baseline chip with a commercial library and the sub-V_t chip with a custom low-voltage library.

	Baseline chip	Sub-V_t chip
Synthesis Conditions	1.1V, slow corner	0.3V, slow corner
Synthesis Clock Period	20ns	200μs
Measured active power (1MHz)	23.1μW at 1V	9.68μW at 0.6V
Switched capacitance ($C_{sw} = P/(fV_{DD}^2)$)	23.1pF	25.9pF

4.3 Timing Verification for Sub-Threshold Logic

In addition to affecting functionality, process variation also increases delay uncertainty. Like device currents, the propagation delay of a logic gate is exponentially dependent on V_t in sub-threshold. Figure 4-14 plots the normalized delay histograms of a microcontroller logic path (composed of multiple logic gates) at 0.3V and 1.2V. To highlight the difference in the relative variation, both histograms are normalized to the sample mean, showing that the relative variation increases by an order of magnitude as V_{DD} is scaled down to 0.3V.

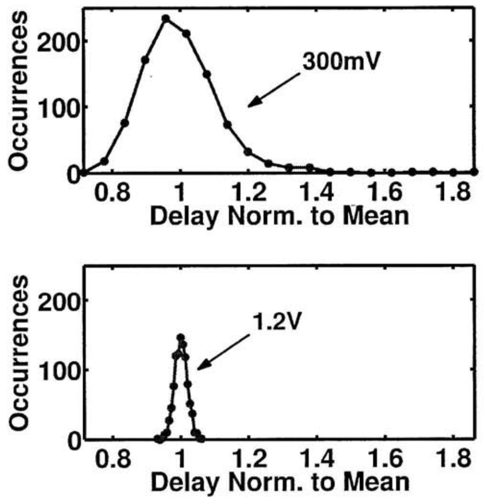


Figure 4-14: Delay histograms of microcontroller logic path ($t_{cq,min} + t_{logic,min}$), each normalized to sample mean to highlight the difference in variability. Both histograms contain 1000 samples.

This large variance in delay is not handled well in conventional library characterization tools used in digital design flows. In a conventional characterization tool, the delays of standard cells are simulated at a range of input signal slew rates and output capacitance. To account for global variation, the characterization is performed at different global process and temperature corners and supply voltages (typically at $V_{DD}\pm10\%$). Until very recently, local variation is often ignored in library characterization. If included, a common method was to model a standard cell as having two delays – a "local-maximum" and "local-minimum" delay – under each set of global process conditions [87]. In the context of sub-threshold, this is akin to representing the delay distribution of a cell by two points, often at fixed percentages above and below the mean delay. More recently, new tools have features to model the distribution of cell delay due to local variation. However, our most recent attempt to apply this at low voltages was unsuccessful as the tool was not able to characterize a flip-flop at 0.5V.

Similarly, conventional tools for timing verification do not function well in sub-threshold. Timing verification consists of checking the setup and hold time constraints for every timing path between two sequential elements or input/output ports (Figure 4-15). The setup and hold time constraints are defined respectively as [50]:

$$t_{c-q,max} + t_{logic,max} + t_{setup} \leq T_{clock} + (t_{clk2} - t_{clk1}) \tag{4.1}$$

$$t_{c-q,min} + t_{logic,min} > (t_{clk2} - t_{clk1}) + t_{hold} \tag{4.2}$$

where

- t_{c-q} is the clock-to-Q delay of a flip-flop
- T_{clock} is the clock period
- $(t_{clk2} - t_{clk1})$ denotes clock skew
- t_{setup} and t_{hold} are the setup and hold time properties of a flip-flop

The constraints are simple to evaluate when cell delays are considered deterministic, as in conventional library characterization. However, as seen previously, cell delays in sub-threshold exhibit substantial variation, which should be accounted for in timing verification.

Accounting for local variation accurately in sub-threshold is complicated by several factors. Like device currents, cell delays exhibit a lognormal distribution in sub-threshold and Gaussian distribution in above-threshold. However, there is no closed form expression for adding lognormally distributed gate delays to obtain the logic path delay [88]. Instead, this must be done with iterative approaches [89] or analytical models, one example being the expression for the sum of identically distributed sub-V_t gate delays in [80]. Further, the setup and hold time properties of flip-flops (t_{setup} and t_{hold} in Equation 4.1 and Equation

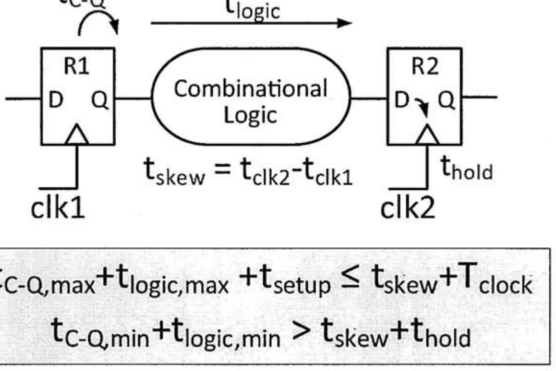

$t_{C\text{-}Q}$
t_{logic}
R1
D Q
Combinational Logic
R2
D Q
$t_{skew} = t_{clk2} - t_{clk1}$
t_{hold}
clk1
clk2
$t_{C\text{-}Q,max} + t_{logic,max} + t_{setup} \le t_{skew} + T_{clock}$
$t_{C\text{-}Q,min} + t_{logic,min} > t_{skew} + t_{hold}$

Figure 4-15: Setup and hold time constraints in a digital logic path.

4.2) are not well-modeled by either Gaussian or lognormal distributions, as illustrated in Figure 4-16.

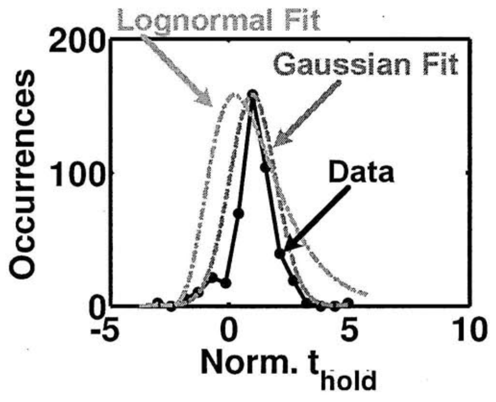


Figure 4-16: Histogram of t_{hold} for a flip-flop at 0.3V.

4.3.1 Variation-Aware Hold Time Verification

A timing approach was developed for verifying hold time constraints Equation 4.2 in the 0.3V microcontroller. In this work we focused on hold time because unlike setup time violations, hold time violations cause race conditions that cannot be fixed by slowing down the clock frequency. The complications described previously make an exact analytical approach difficult. In this design we employed an approach based on Monte Carlo simulation to accurately capture the hold time constraint on critical paths, while using analytical methods to reduce the total simulation effort.

Since performing Monte Carlo simulations on all timing paths in a circuit would be prohibitively time-consuming, the basic idea here is to pick a subset of paths that are more likely to violate hold time constraints and simulate them in detail. Figure 4-17 gives an overview of the timing flow.

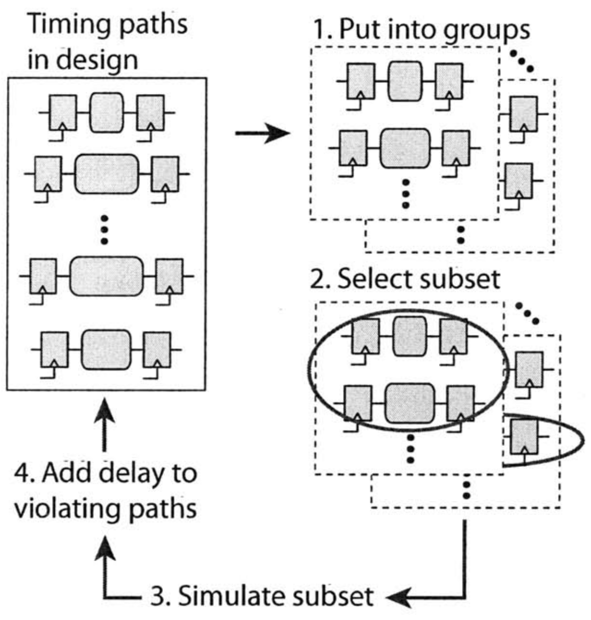


Figure 4-17: Overview of the proposed hold time verification flow.

First, we obtain a list of timing paths in the design after it has been placed and routed. For a small design such as the 0.3V microcontroller, this can be an exhaustive list. For a large design, the list can leave out timing paths with very large timing margins. In the context of hold time violations, the hold time margin is defined simply by rearranging Equation 4.2:

$$t_{hold-margin} = t_{c-q,min} + t_{logic,min} - (t_{clk2} - t_{clk1}) - t_{hold} \tag{4.3}$$

The hold time margin must be non-negative for proper functionality. In other words, a timing path with a long logic delay relative to clock skew has a large margin and is thus unlikely to violate hold time.

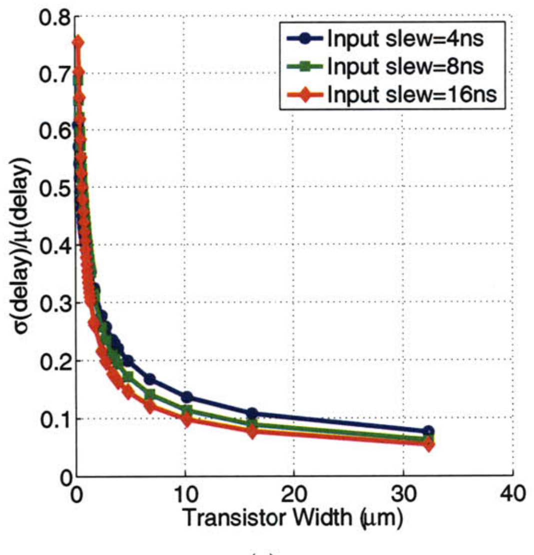
The list of timing paths is generated under the worst case global conditions – at the fast process corner for verifying hold time, and at low V_{DD} where V_t variation is most prominent. However, the list does not consider local variation. Known paths with very short logic delays (e.g. shift registers) are removed from the timing report and handled separately.

The timing paths are put into groups according to their mean hold time margin (i.e. $t_{hold-margin}$ reported by commercial timing tools, without local variation). At this point, the place and route tool should have ensured all paths have positive mean hold time margin. A subset of paths in each group, which we will call the critical paths, is then selected for Monte Carlo simulation. More than one critical path is simulated per group because the path selection makes several approximations to speed up the analysis. The paths are split up into groups so that if Monte Carlo simulations reveal that the critical paths are likely to violate hold time, fixes can be applied to the entire group.

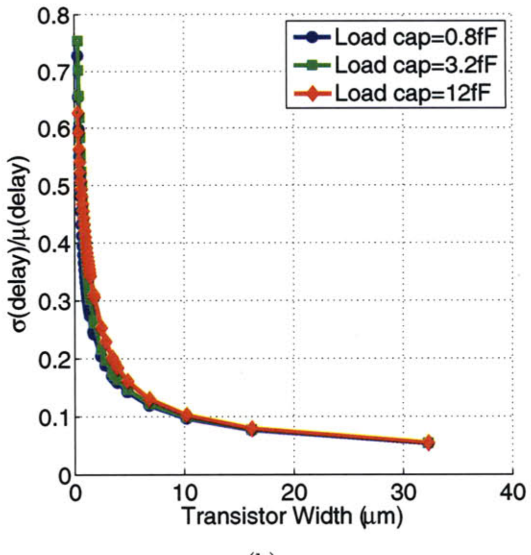
The path selection aims to find paths that have large standard deviation over mean hold time margin. Since the paths in a group have similar mean hold margins, this is approximately the same as finding the paths with the largest standard deviation or variance. There are two main approximations made in the path selection, and we guard against the inaccuracies due to these approximations as will be described later. First, to estimate the variance of the path, we first assume that the stages that make up the path (i.e. delays of individual gates) are uncorrelated. For conciseness, we will refer to these as the path variance and gate variance respectively. The above assumption allows us to find the path variance by adding the variance in delay of the individual logic gates. Second, we model gate variance as a function of the size of the transistor switching the output, the rise/fall times of the input, and the load capacitance at the output of the gate. This is done instead of characterizing the delay variance of every single logic gate in a library across different input/output conditions, which would be extremely time-consuming especially for large libraries.

We capture how gate variance changes with transistor sizes, input rise/fall times, and output load capacitance in a series of lookup tables that can be precharacterized for the process technology. Figure 4-18 plots partial results from this characterization, showing $\sigma_{delay}/\mu_{delay}$ versus transistor width, performed over a range of input slews (rise/fall times) and load capacitances. It is observed that $\sigma_{delay}/\mu_{delay}$ does not change much with load capacitance and decreases slightly for inputs with slow rise/fall times.

The path selection is validated by perfoming Monte Carlo simulations on all the paths in a group, and seeing which paths were identified by the path selection approach. Figure 4-19 illustrates the mean and standard deviation of all the paths in one group found by



(a)



(b)

Figure 4-18: $\sigma_{delay}/\mu_{delay}$ versus transistor width in an inverter, taken at different input rise/fall times and load capacitances.

Monte Carlo simulation. The paths selected by the proposed approach are identified with red circles, and the remaining are identified by blue squares. It is seen that the approach can identify the critical paths (i.e. the ones with the largest standard deviation over mean).

It should be noted that the original assumption of uncorrelatedness may not hold, since the delay of one logic gate depends on the output rise and fall times of the logic gate which precedes it in the timing path. Therefore, the estimated path variance may not be fully accurate. To address this, we select multiple paths within each group for simulation.

The selected paths from all the groups undergo Monte Carlo simulation with local variation and at the global fast corner, giving an accurate histogram of their $t_{hold-margin}$, with an example shown in Figure 4-20. A sample in the Monte Carlo simulation is considered to violate hold time when $t_{hold-margin} < 0$. Since the probability of $P(t_{hold-margin} < 0)$ is typically small (less than 1%), finding it accurately from the raw data alone would require an impractically large number of simulations. Therefore, we fit the data to a Gaussian distribution and then estimate $P(t_{hold-margin} < 0)$. A Gaussian instead of lognormal distribution was used because the latter is undefined for negative values.

If $P(t_{hold-margin} < 0)$ is above a set threshold, as determined by the number of paths in the design and the desired timing yield, then extra delay buffers are inserted in the logic path to increase $t_{logic,min}$ and the hold time margin. Since the simulated paths are

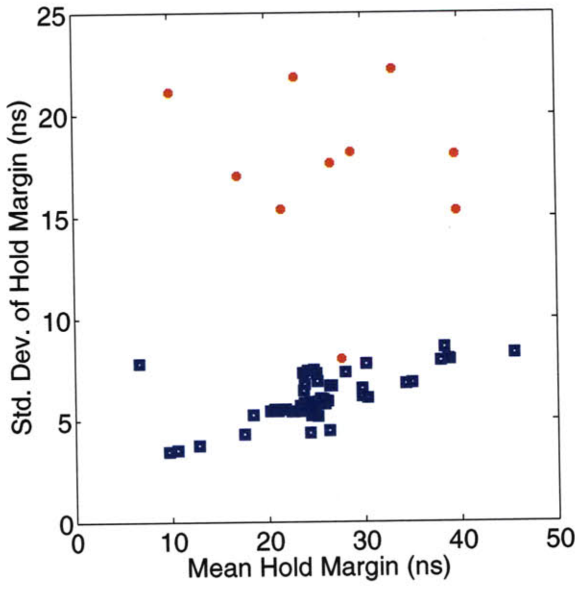


Figure 4-19: Validating the path selection approach. Scatter plot shows the mean and standard deviation of the timing paths that were placed in one group in the timing flow. The path selection correctly identifies the critical paths with the largest standard deviations (most likely to fail).

the critical paths from a group, buffers are also applied to other (unsimulated) paths in the same group. In the 0.3V microcontroller design, paths requiring extra buffering were concentrated in small groups with low average hold time margin. As a result, 151 paths were fixed in the hold time verification flow.

It should be noted that a variation-aware approach typically results in fewer delay buffers inserted compared to worst case timing analysis. For instance, a common worst case methodology uses two deterministic values to model fast and slow delay in a cell under local variation. One such example would be to use the $\pm 1\sigma$ points as the slow and fast delays. Hold time constraint is verified by assuming all cells in the data path have fast delays, while those in the capture clock path have slow delays, in order to obtain the worst case scenario. However, in reality, it is unlikely that all cells in the data path uniformly exhibit fast delay due to local variation. Because of this pessimism, the worst case methodology identified 929 timing paths for hold time fixing, several times more than the 151 paths selected by the variation-aware approach.

The tools used and the run time of each step of the flow is reported in Table 4.2. The run time is measured on a Linux workstation with a 2.3GHz quad-core CPU.

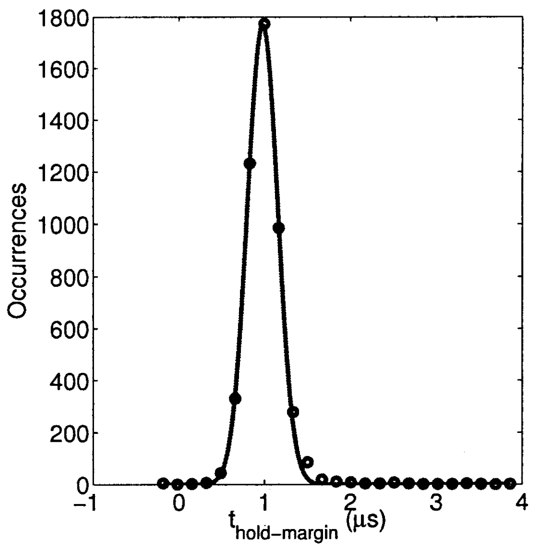


Figure 4-20: Example histogram of $t_{hold-margin}$ of one timing path in the 0.3V microcontroller. Points show simulated data and line shows a fitted Gaussian distribution.

Table 4.2: Tools used in each step of the timing flow, and the associated run time on a 2.3GHz CPU.

Step	Tool	Run Time
Path selection	MATLAB	20 minutes
Simulation of one selected path (1k-point Monte Carlo)	SPICE	5 minutes
Analyzing simulation results	MATLAB	several minutes

Accuracy Versus Run-Time Trade-Offs

There are several ways in which the accuracy of this timing flow can be increased at the expense of longer run time as discussed below.

- Account for the correlation between the stages in a path. For this we would need to characterize the covariances of all pairs of logic gates in a path. This characterization time grows quadratically as the number of logic gates in a library.

- More extensive precharacterization of the gate variance. This includes simulating over more input and output conditions, as well as characterizing the delay variance of stacks of transistors.

- Perform a longer Monte Carlo simulation of each path to get a more accurate hold time margin distribution for the path.

4.3.2 Comprehensive Delay Variation Data

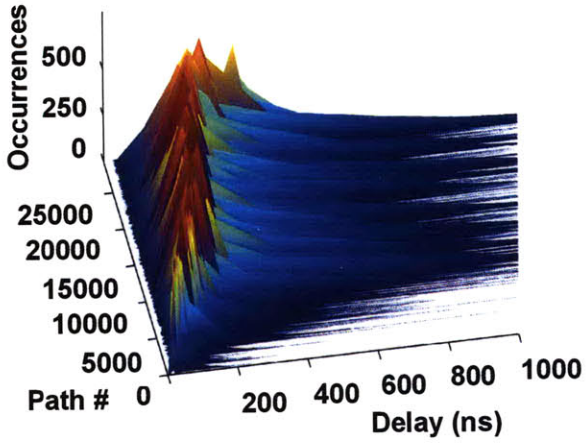
Apart from the analysis described above, we performed Monte Carlo SPICE simulations over several months on 30000 timing paths in the microcontroller. A 1k-point simulation was performed for each path which requires approximately 5 minutes. The results serve to illustrate trends in sub-V_t delay variation.

In Figure 4-21(a), each horizontal cross section is the logic delay histogram of one timing path under local variation, at 0.3V and global fast corner. The rightward skew is typical of a lognormal distribution. Figure 4-21(b) shows a scatter plot of the corresponding timing path statistics. Each point represents one path, with mean delay plotted on the x-axis and σ/μ shown on the y-axis. Initially, the lower range of σ/μ decreases with the mean delay, which reflects how variation tends to average out in longer paths. However, this quickly reaches diminishing returns, and σ/μ does not decrease far below 0.1, even for very long paths. The same trend is observed when logic depth (the number of stages in the path), instead of mean delay, is plotted on the x-axis. Since σ/μ depends on both device sizes and logic depth, the lower bound observed reflects the inherent variability given the device sizes used in the standard cell library. Additionally, the upper range indicates that outliers with large amounts of variation occur less frequently as the path length increases. However, it is important to note that the shortest path is not necessarily most likely to violate hold time, because slightly longer paths can exhibit significantly higher variability.

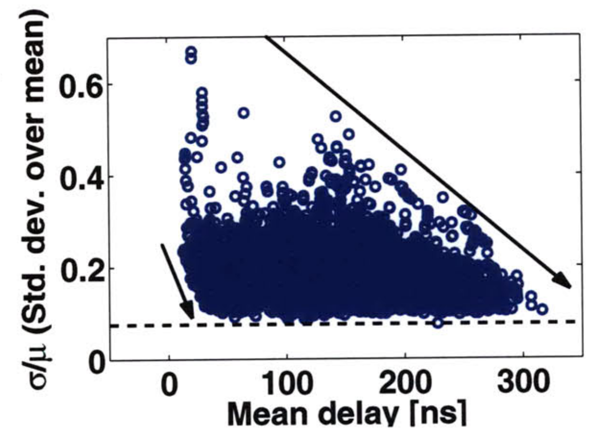
4.4 Standard Cell Library for Voltage-Scalable Logic

The previous sections discussed standard cell and timing verification for logic targeting minimum-energy operation in sub-threshold. In the next two sections, we address the two topics for voltage-scalable logic operating from near-threshold up to nominal V_{DD} to meet more stringent speed constraints. The biomedical processing platform is one such example.

While designing standard cells to function in sub-threshold is challenging, circuits become less sensitive to local variation as V_{DD} is increased. Therefore, at voltages close to the the transistor threshold, a conventional, above-threshold cell library may be sufficient. Nevertheless, it is important to verify functionality of the library with Monte Carlo simulations at the lowest targeted V_{DD} and at the worst case global corners. Section 4.2.2 has noted that the cells most susceptible to local variation are sequential elements (i.e. flip-flops



(a)



(b)

Figure 4-21: (a) Delay histograms of 30k microcontroller timing paths at 0.3V, fast corner. Each horizontal cross section represents distribution of one path. (b) Scatter plot of microcontroller timing path statistics corresponding to data in (a).

and latches) and logic gates with many parallel leaking devices fighting series-connected devices (i.e. high fan-in NAND/NOR gates). For example, the biomedical platform in this thesis aims to operate at 0.5V, slightly above V_t in the $0.13\mu m$ process, and is able to use a conventional cell library.

4.5 Design Flow and Timing Verification for Voltage-Scalable Logic

In designing a logic circuit to function across a wide voltage range, one key consideration is that the propagation delays of logic gates do not scale in the same manner across V_{DD}.

Unfortunately, the digital design flow and common design tools are targeted towards circuits that operate within a narrow range of supply voltages. In this section we describe issues due to non-uniform delay scaling and show a modified design flow to address them.

The basic digital design flow is illustrated in Figure 4-22, and we will discuss issues related to several steps in the flow: synthesis, place and route, and hold time fixing. The synthesis and place and route steps employ a pair of timing libraries characterized at two closely spaced voltages, forming a guard-band around a nominal value (in this case, $V_{DD} \pm 10\%$) in order to account for voltage supply noise. Accordingly, in designing a circuit that scales across a wide voltage range, we must select one nominal V_{DD} for synthesis and place and route. A natural choice is to design the circuits at the highest V_{DD} so that the circuit can be optimized to achieve the desired maximum speed. In other words, the setup time constraint which sets the maximum frequency is more important at the high voltage end. However, as we will show next, the hold time constraint is more critical at the low voltage end.

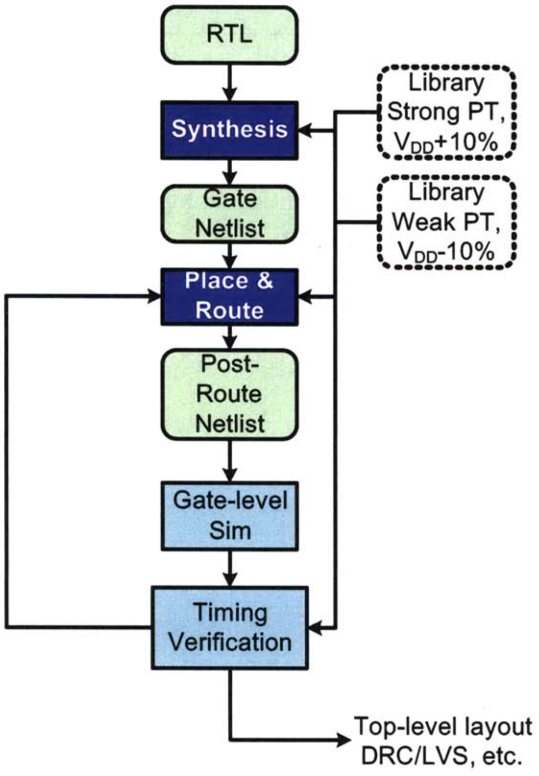


Figure 4-22: Typical digital design flow. Issues related to voltage scaling in the dark shaded steps will be discussed in this section.

In the following, we synthesize the biomedical platform at high V_{DD} and find the hold time margin (as defined in Equation 4.3) across its timing paths at three V_{DD} within its operating range: {0.55V, 0.7V, 1V}+10% [2]. Figure 4-23 shows a histogram of the hold time margin across 10,000 timing paths at the three voltages. The majority of samples were close to 0, so the x-axis is "zoomed in" to show details where the hold time margin is less than 5% of the clock period. The y-axis shows the proportion of timing paths with timing margins corresponding to the x-axis value.

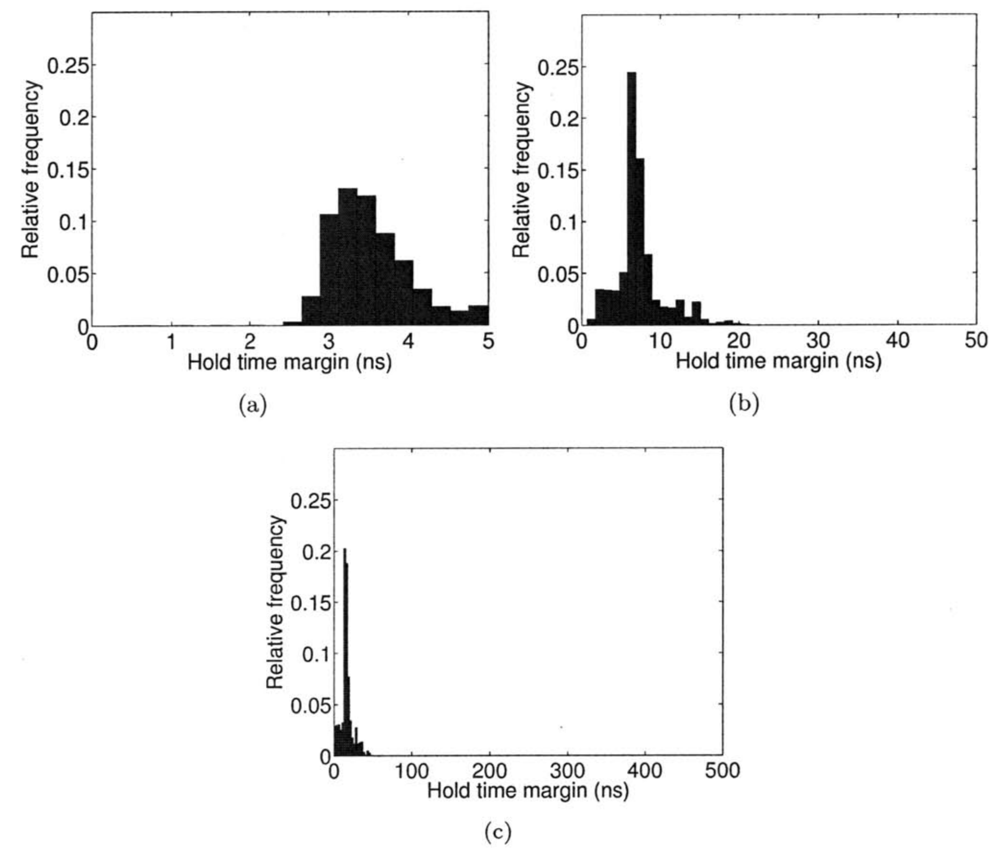


Figure 4-23: Histograms of the hold time margin in timing paths of the biomedical platform. Data taken at (a) V_{DD}=1.1V, (b) 0.77V, and (c) 0.6V. The x-axis scale is set to 5% of the clock period. Relative frequency refers to the proportion of timing paths with hold time margins corresponding to the x-axis value.

The data shows that the design has adequate hold time margins at 1.1V because the design flow explicitly optimized the design at this voltage. However, the hold time margin, as a fraction of the clock period, degrades at lower voltages, as reflected by the histogram shifting to the left and clustering close to 0. From this we can conclude that hold time

[2]Note this was done with Synopsys Primetime, a conventional timing tool, and does not include local variation.

margins do not scale proportionally with the maximum path delay as V_{DD} is decreased. Moreover, after implementing a design at one voltage, it is imperative to check timing constraints at other voltages in the operating range and fix any violations. In addition, the variation-aware timing approach described in Section 4.3 can be used for additional verification at the low voltage end. We modified the digital flow accordingly for use in designing the voltage-scalable biomedical platform. The modified flow is shown in Figure 4-24.

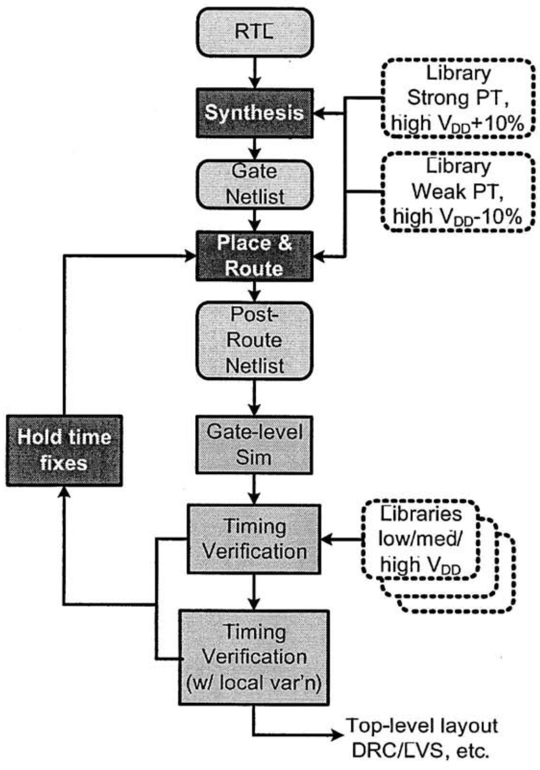

RTL
Synthesis
Library Strong PT, high V_DD+10%
Gate Netlist
Library Weak PT, high V_DD-10%
Place & Route
Post-Route Netlist
Hold time fixes
Gate-level Sim
Timing Verification
Libraries low/med/ high V_DD
Timing Verification (w/ local var'n)
Top-level layout DRC/LVS, etc.

Figure 4-24: Digital flow for voltage-scalable circuits. *PT* refers to process and temperature.

The non-uniform scaling of logic gate delays with V_{DD} also has implications for the special delay cells used to fix hold time violations. Recall that the hold time margin can be increased by slowing down the logic path. Typically this is done by extending the logic path with special delay cells, specifically designed to introduce long delays, by employing longer-than-minimum length transistors. Unfortunately, due to reverse short channel effects [90] in modern process technologies, the drain current at low V_{DD} actually goes up with increasing device lengths. Therefore, delay cells become less effective at slowing down the logic path with voltage scaling. The difference in delay scaling between a delay cell and a

NAND gate is plotted in Figure 4-25.

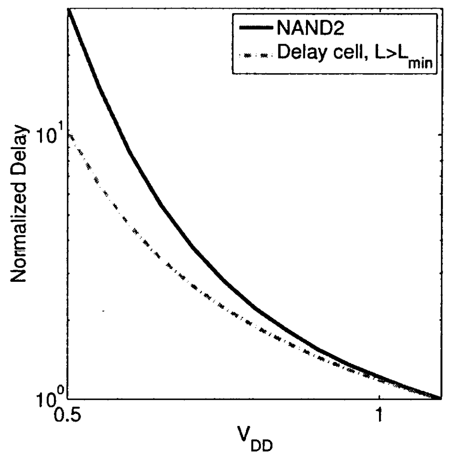
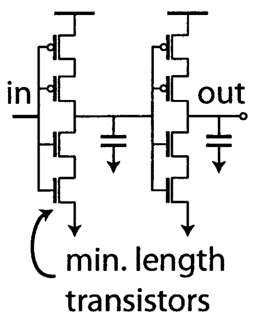


Figure 4-25: Delay scaling across V_{DD} of 2-input NAND gate and typical delay cell. Both curves are normalized to the minimum delay of the cell. Due to reverse short channel effects, the delay cell becomes less effective at low V_{DD}.

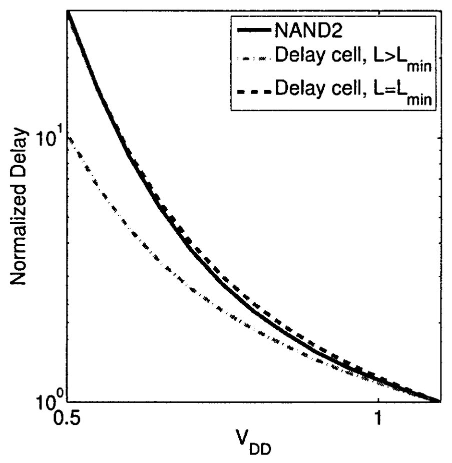
When such delay cells are used in a voltage-scalable design, the hold time margin at low V_{DD} significantly worsens. The first row of Table 4.3 lists the negative hold time margin at 0.5V, summed across all violating paths in the biomedical platform if conventional delay cells were used. The large negative margin indicates many violations that must be fixed manually by inserting an impractically large number of cells. Clearly this is not feasible, and instead we employed an alternate delay cell with only minimum length devices. We increase the delay provided by this alternate cell by adding small MOS capacitors within the cell as illustrated in Figure 4-26(a). This alternate cell has a larger area than the conventional cell, but allows hold violations to be corrected with much less power. As shown in Figure 4-26(b) and Table 4.3, this alternate cell maintains effectiveness at low V_{DD} and greatly reduces the amount of hold time violation in the biomedical platform, which are then fixed in a subsequent design iteration.

Table 4.3: Negative $t_{hold-margin}$ summed across all violating paths in the biomedical platform. The first design uses conventional delay cells, resulting in severe hold violations at 0.5V. The use of alternate delay cells greatly reduces the amount of hold time violations.

Delay cell Type	Number of Violating Paths at 0.5V	Total Negative $t_{hold-margin}$ at 0.5V
Conventional ($L > L_{min}$)	174	-49094ns
Alternate ($L = L_{min}$)	100	-117ns



(a)



(b)

Figure 4-26: (a) Alternate delay cell that maintains effectiveness at low V_{DD}. (b) Delay scaling across V_{DD} of NAND gate, typical delay cell ($L > L_{min}$), and alternate delay cell ($L = L_{min}$). The delay of the alternate cell scales approximately in the same way as NAND2.

4.6 Ultra-Low-Voltage Microcontroller

A microcontroller system-on-chip, designed to operate down to 0.3V, demonstrated the logic design and timing methodology described in this chapter. While previous work has shown the energy savings afforded by ultra-low-voltage operation, most of these systems were designed in older process technologies where local variation is less prominent. For example, a 180mV, $0.18\mu m$ FFT processor was presented in [81], while a $0.13\mu m$ processor with 8-bit ALU, 32-bit accumulator, and a 2kb SRAM functional down to 200mV was implemented in [25]. Body biasing and several gate sizing strategies were examined in a $0.13\mu m$ sub-V_t processor [24].

Looking forward, technology scaling enables reduced CV_{DD}^2 energy and increased density, but presents a new challenge in the form of heightened intra-die variation. In [91], authors presented a 65nm 320mV motion estimation accelerator to compute the sum of absolute difference (SAD) between a pair of 8 pixels. The chip employed optimized datapath circuits to address weak I_{ON}/I_{OFF} ratio and threshold voltage variation. For instance, registers contained fully-interrupted, upsized keepers, and multiplexers with more than 3 inputs were remapped into 2:1 multiplexers. In [92], a 65nm SRAM design with a 10T bit-cell achieved functionality down to 400mV.

The work of [91] and [92] involve regular circuit structures that can be hand-crafted to ensure functionality down to low voltages. In this thesis, the microcontroller was synthesized and laid out with a standard digital design flow, and hence its functionality relies heavily on logic design and timing methodology. The demonstrated functionality shows that it is feasible to synthesize sub-threshold digital circuits in advanced process technologies despite significant process variation.

We also note that soft digital signal processing is an interesting approach for a specific class of applications where a small amount of computation error (e.g. due to timing errors resulting from local variation) are acceptable and are considered as noise in the system. For instance, computation errors in an FIR filter can be considered as a degradation of the output signal-to-noise ratio. In [93], the authors propose exploiting this for energy savings by lowering the V_{DD} of a DSP below a critical voltage, beyond which timing errors start to occur for certain inputs that exercise the critical path. An error control block of low complexity monitors the output of the DSP. A test-chip in [94] demonstrates this concept. The idea of soft DSPs has been subsequently extended in the literature, for example in [95, 96]. While these approaches are attractive for datapath circuits, it is important to note that control circuits are often much less tolerant to errors, and proper design methodologies are still critical in those cases.

4.6.1 System Overview

The microcontroller logic was part of a system-on-chip which also included a custom low-voltage SRAM [97] designed by Naveen Verma and a switched capacitor DC-DC converter [98] designed by Yogesh Ramadass. As pictured in Figure 4-27, both logic and SRAM were designed to operate between 0.3V to 0.6V. To realize the full energy savings from voltage scaling, a DC-DC converter which efficiently converts a battery voltage to low voltage and power levels is crucial. Further, minimizing the number of external components is desirable in embedded applications. Accordingly, the system features a DC-DC converter which is fully integrated on chip, and can provide variable voltages at microwatt power levels with high efficiency.

Figure 4-28 shows a block diagram of the core logic, which is based on the MSP430 microcontroller architecture [68]. The 16-bit Reduced Instruction Set Computer (RISC) CPU supports 27 instructions and 7 addressing modes of the standard MSP430 instruction set. The microcontroller interfaces to 128kb of unified instruction and data memory, imple-

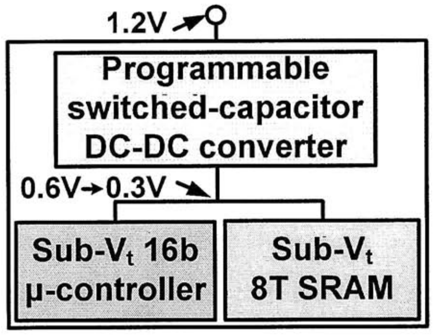
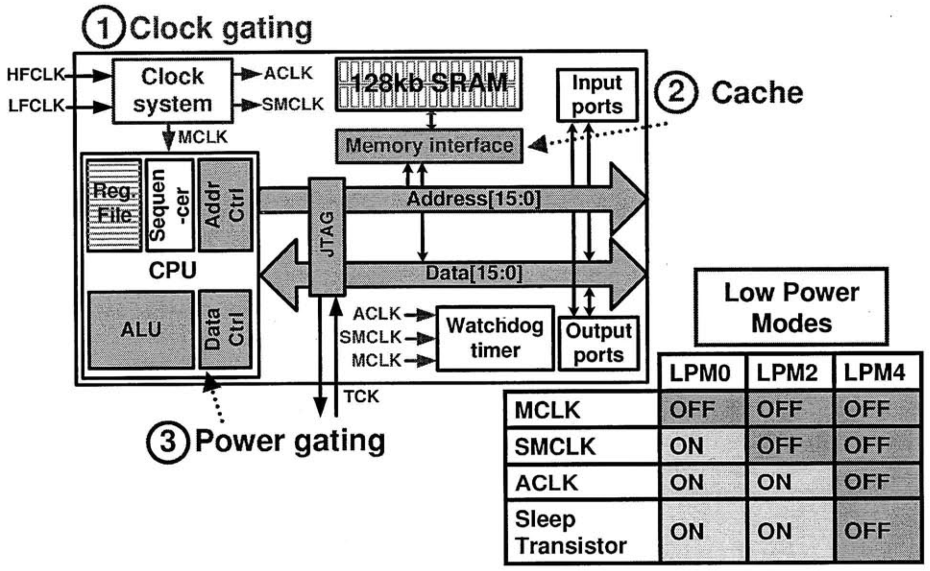

1.2V
Programmable switched-capacitor DC-DC converter
0.6V→0.3V
Sub-V_t 16b µ-controller
Sub-V_t 8T SRAM

Figure 4-27: Block diagram of the system-on-chip.

mented as a custom SRAM, as well as to a watchdog timer and general purpose I/O ports. Programming of the SRAM is performed at startup via a JTAG interface.



	LPM0	LPM2	LPM4
MCLK	OFF	OFF	OFF
SMCLK	ON	OFF	OFF
ACLK	ON	ON	OFF
Sleep Transistor	ON	ON	OFF

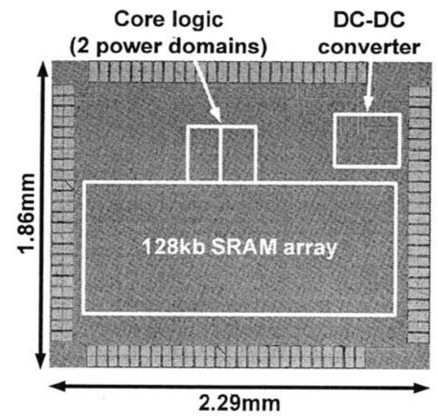
Figure 4-28: Block diagram of microcontroller core.

Targeting low power applications, the microcontroller provides several power management features as illustrated in Figure 4-28. The clock system, which distributes external clocks to the microcontroller logic, supports three low power modes. In the first mode (LPM0), the master clock (MCLK) going to the CPU is gated. At this time, the CPU does not perform any processing, although peripherals remain active. The high frequency clock for the peripherals, or the sub-system master clock (SMCLK), is disabled in the second low power mode (LPM2). However, the auxiliary clock (ACLK), the low frequency clock for peripherals, remains on so that peripherals can function with lower active power. In the standby mode (LPM4), all clocks are shut off. The microcontroller can wake up from any of these modes through an interrupt event generated by the watchdog timer or input port.

This implementation also contains two features not found in commercial versions of the MSP430 microcontroller. First, the memory interface contains a small cache to reduce the memory access power. One 64-bit row of memory, which contains four 16-bit CPU words, is fetched and stored at a time. Successive 16-bit accesses to the same row require no further memory activity. This provides up to 50% savings in the measured memory access power for applications with a high hit rate. Second, the logic is split into two power domains; the unused blocks shaded in Figure 4-28 are power gated during standby mode. Key CPU states are retained such that the microcontroller can continue program execution upon emerging from standby. The on-chip sleep transistor is sized for approximately 5% delay penalty at V_{DD}=300mV. Accounting for the energy overhead in turning this transistor on and off, the breakeven time for power gating is less than 100μs.

4.6.2 Prototype Measurements

A summary and die micrograph of the test chip, fabricated in 65nm CMOS, is shown in Figure 4-29. The DC-DC converter, including charge transfer capacitors, occupies just 0.12mm^2. The minimum energy point of the microcontroller occurs at 500mV, and functionality was verified down to 300mV.



(a)

Process	65nm CMOS
Area	
DC-DC Converter	0.12mm^2
SRAM	1.36mm^2
Logic	0.14mm^2
Performance	µ-controller (Logic and SRAM)
Minimum Energy Point	500mV
Minimum Functional V_{DD}	300mV

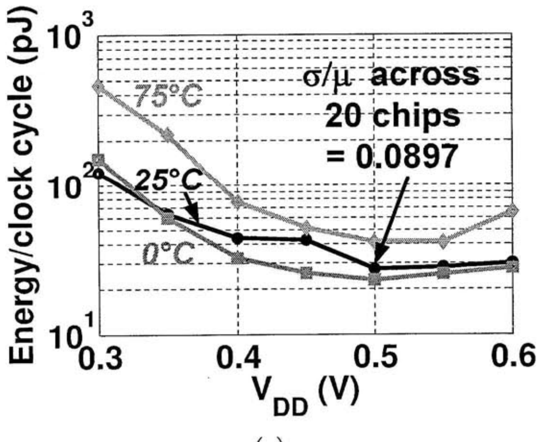
(b)

Figure 4-29: (a) Die micrograph and (b) summary of microcontroller test chip.

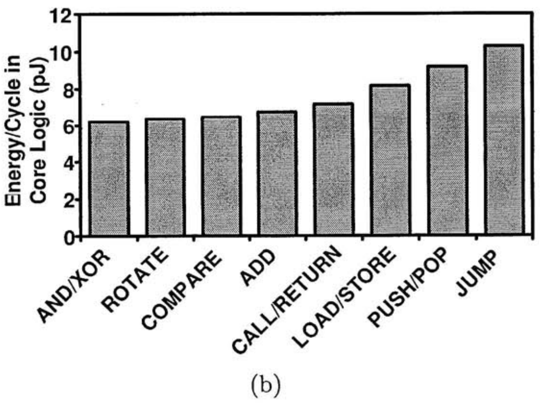
Active Energy and Performance

Figure 4-30(a) plots the measured energy per cycle versus supply voltage for the microcontroller logic and SRAM at 0^oC, 25^oC, and 75^oC. The energy is measured while the system executes test code which cycles through the available instructions and addressing modes. Since the I/O pads, logic, and memory array are operated at the same voltage, level shifters are not required on-chip. Level converters are used on the test board to interface the low-voltage I/Os to the logic analyzer. Memory and logic together consume 27.2pJ per clock cycle at 500mV and 25^oC. The optimum energy does not vary much across 20 chips; the measurements have a σ/μ of 0.0897.

Shown in Figure 4-30(b) is the energy consumption of the microcontroller core logic while it executes specific instructions. Generally, instructions for arithmetic or boolean operations (e.g. add, and, compare), executed on operands stored in CPU registers, require roughly the same amount of energy per cycle. Instructions that involve memory accesses for data (e.g. load/store, push/pop) exhibit higher energy consumption as expected. The jump instruction, which generates high switching activity on the address bus, requires the most energy.



(a)



(b)

Figure 4-30: (a) Energy versus V_{DD} of logic and memory over temperature. The σ/μ of measurements across 20 chips at 500mV is shown. (b) Energy of microcontroller core logic while it executes different instructions at 500mV, room temperature.

The energy consumed by the SRAM array per system clock cycle is shown in Figure 4-31. The memory greatly influences the minimum energy point of the system since it consumes a major portion of the total system energy, highlighting the importance of reducing memory energy through voltage scaling and other circuit techniques.

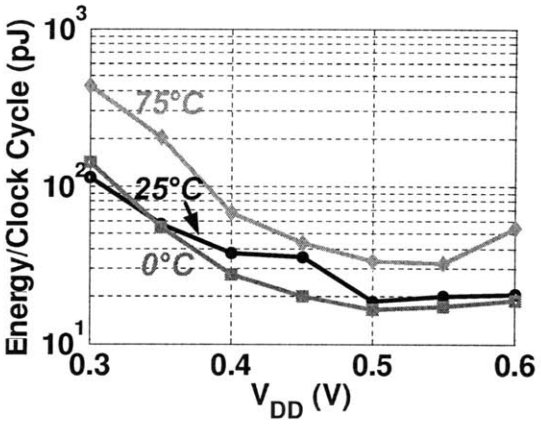


Figure 4-31: Energy versus V_{DD} of the SRAM array per system clock cycle.

The efficiency of the DC-DC converter delivering 500mV is shown in Figure 4-32. The converter achieves more than 75% efficiency with an order of magnitude change in load power, between $10\mu W$ to $250\mu W$. With the microcontroller as a load, the converter provides 75% efficiency at $12\mu W$. When measured standalone, the converter reaches a peak efficiency of 78%.

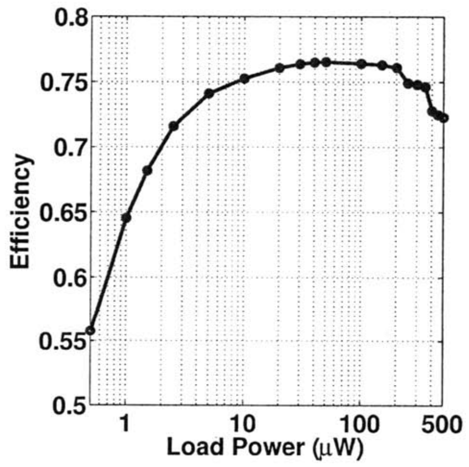


Figure 4-32: DC-DC converter efficiency while delivering 500mV. The converter is powered by a 1.2V supply.

Figure 4-33 plots the microcontroller performance versus supply voltage at 0^oC, 25^oC, and 75^oC. The measured frequency, accounting for logic and memory delays, is 434kHz at 25^oC and 500mV. The frequency ranges from 8.7kHz to 1MHz across the operating range of 0.3V to 0.6V. The σ/μ of measurements across 20 chips at 500mV is 0.133.

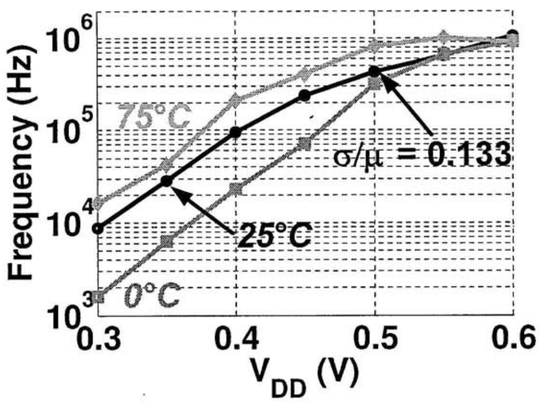


Figure 4-33: Frequency versus V_{DD} across temperature. The σ/μ of measurements across 20 chips at 500mV is shown.

Standby Power

The inclusion of a DC-DC converter enables the system to dynamically scale V_{DD} to 300mV during standby mode, where memory and logic together consume less than $1\mu W$, as shown in Figure 4-34. Accounting for the DC-DC converter efficiency loss at such low power levels, this represents a $2.1\times$ reduction in leakage power compared to keeping V_{DD} constant at 500mV during standby.

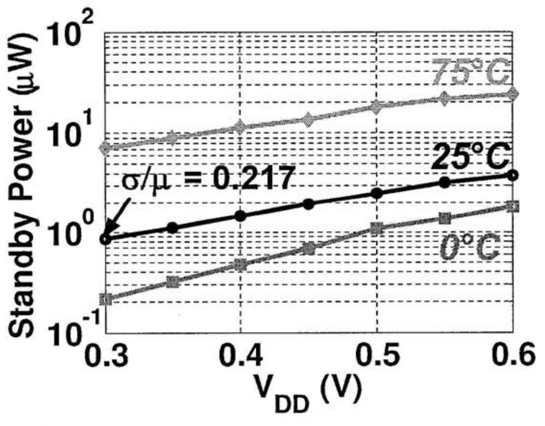


Figure 4-34: Standby power versus V_{DD} across temperature. The σ/μ of measurements across 20 chips at 300mV is shown.

4.7 Conclusion

Process technology, of course, has deep implications for the design of low-voltage digital circuits. We first examine the movement of the minimum energy point with process scaling with predictive models at the 65nm, 32nm, and 22nm nodes. For a 32-bit adder characterization circuit, the optimum V_{DD} increases with process scaling, but still remains in the sub-threshold region, motivating the design of sub-threshold circuits. Since recent technology nodes embody a trade-off between lower active energy and higher speed versus lower leakage power, we consider the issue of selecting a technology given the frequency constraint and duty cycle of an application. At low duty cycles when leakage energy becomes significant, an older technology node can result in lower total energy. This approach was used in selecting an appropriate process flavor for the biomedical processor.

This chapter considered logic design issues in two classes of low-voltage circuits: one targeted for deep sub-threshold operation at the minimum energy point, and the other operating from near transistor threshold to nominal V_{DD}, to support commensurate performance requirements. The former class of systems is primarily concerned with achieving functionality despite prominent process variation. We analyze the area and switched capacitance overhead of designing a 65nm standard cell library to operate at 0.3V. Two microcontrollers with the same functionality were fabricated, one with a conventional library and one with the 0.3V sub-V_t library. Chip measurements show that switched capacitance overhead of the sub-V_t library was roughly 12%, but the library enables energy reduction overall by allowing aggressive voltage scaling.

Circuits operating in sub-V_t exhibit order-of-magnitude higher delay variation than seen at above-V_t. This renders conventional timing verification tools inaccurate. Instead, we present an approach to verify timing in a microcontroller using Monte Carlo simulation for accuracy but selecting critical paths analytically to reduce simulation time.

The sub-V_t library and timing approach were demonstrated in a 65nm MSP430-based microcontroller operating down to 0.3V. This represents the first processor to achieve deep sub-threshold operation at the 65nm node, and the first to incorporate standard cell design and timing methodology explicitly considering local variation.

Chapter 5

Low-Voltage SRAM

In many systems ranging from embedded microcontrollers to high performance desktop CPUs, the memory often occupies a dominant portion of the area and power. Similarly, memory is a key component in the processing platform, being responsible for storing program instructions and data. In this type of embedded processor, a static random access memory (SRAM) is a natural choice for storing content that is frequently accessed – it supports random access unlike flash memory, does not require specialized process technology unlike embedded DRAM, and can be operated at lower voltage and power than Ferroelectric RAM (FeRAM). On the other hand, non-volatile memories such as flash or FeRAM are suitable for storing data that must be retained for long time periods even if the processing platform is powered off. In this chapter we focus on the design of an SRAM as the primary storage for instructions and short-term data; in a system prototype, data requiring non-volatile storage would be written out into external flash memories.

Since the SRAM is involved in the vast majority of activities on the processor – providing instructions, storing processed results – reducing its energy is crucial. Further, it would be desirable to operate the SRAM at the same voltage as the logic to avoid the need for level conversion, which imposes delay and power overhead. Conventional SRAM circuits for nominal-voltage operation are optimized for high density and performance at a cost of reduced robustness. This trade-off works admirably at nominal V_{DD} but quickly breaks down at low voltages where circuits become more sensitive to process variation. Accordingly, researchers have explored a range of bit-cells, peripheral circuits, and architectures to enable low-voltage operation. A detailed discussion of prior work as well as a low-voltage SRAM

Table 5.1: Related work in low-voltage SRAMs.

Reference	Process Technology	Key Features
Takeda, ISSCC 2005 [100]	90nm	7T bit-cell with 7^{th} transistor added to prevent read upset condition
Calhoun, ISSCC 2006 [101]	65nm	10T bit-cell, floating VV_{DD} to aid writing
Chen, JSSC 2006 [102]	$0.13\mu m$	Register file-based cell; multiplexed read scheme; self-timed keepers
Kim, ISSCC 2007 [103]	$0.13\mu m$	10T bit-cell with data-independent bitline leakage; virtual ground replica scheme to aid reading
Zhai, ISSCC 2007 [104]	$0.13\mu m$	6T bit-cell modified for single-ended read; floating virtual V_{DD} and ground rails to aid writing
Verma, ISSCC 2007 [105]	65nm	8T bit-cell modified to reduce bitline leakage during read; boosted wordline and actively driven VV_{DD} to aid writing
Chang, VLSI 2007 [106]	65nm	8T bit-cell, short read bitlines and long write bitlines; gated diode sense amplifier
Kulkarni, JSSC 2007 [107]	$0.13\mu m$	10T bit-cell with hysteresis to improve read margin
Chang, ISSCC 2008 [108]	90nm	10T bit-cell supporting column-wise selection and column interleaved layout
Sinangil, ESSCIRC 2008 [109]	65nm	Optimized for wide operating range (0.25-1.2V) with reconfigurable read and write assist circuits
Sinangil, A-SSCC 2009 [110]	45nm	8T bit-cell with new array architecture to enable column interleaving; loop to select reference voltage for sense-amplifier

design are given in [99]. We summarize some related work in low-voltage SRAMs and their key features in Table 5.1.

This chapter outlines the challenges of designing low-voltage SRAMs and the first points of failure when V_{DD} is reduced. Next, bit-cell and peripheral circuits enabling low-voltage operation are discussed; the energy of two competing write assist circuits are analyzed and a model for the read current distribution is proposed. Finally, we present a mechanism for decreasing glitch energy on the SRAM data bus which, unlike common practice in SRAM design, considers the average instead of worst case to remove glitches at low leakage cost.

5.1 Low-Voltage SRAM Challenges

Hold and Read Static Noise Margin

The conventional 6-transistor (6T) bit-cell pictured in Figure 5-1 forms the basis of modern SRAM designs, but suffers from several vulnerabilities in the presence of process variation which render it non-functional at low voltages. To achieve high area density, the 6T bit-cell relies on ratioed device sizing to set the relative device strengths needed for functionality. Since sizing changes current linearly while V_t variation has an exponential impact in sub-threshold, variation can easily overwhelm the effect of sizing to cause bit-cell failures. The three pairs of transistors in the 6T cell are each primarily involved in one aspect of the cell operation. In the subsequent discussion, we will refer to these devices as labeled in Figure 5-1 – the NMOS pass-transistors are called the access devices, the NMOS pulling to ground are the driver devices, and the two PMOS are the load devices.

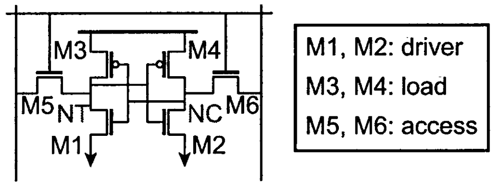


Figure 5-1: Conventional 6-transistor bit-cell

Data retention in a 6T SRAM bit-cell is determined by the cross-coupled inverters M1-M4 in Figure 5-2(a). The ability of a bit-cell to retain data can be characterized by its butterfly plot as illustrated in Figure 5-2. The two bi-stable intersection points in the butterfly plot, shown by red circles in Figure 5-2(a), indicate that the bit-cell can support "0" and "1" logic levels, and thus proper data retention. The static noise margin (SNM) indicates the maximum amount of noise that can be applied to the storage nodes of the bit-cell before the state of the cell is destroyed. The SNM is measured as the edge length of the largest inscribed square in the butterfly plot [111]. If variation causes both VTCs to be shifted by more than this amount, the butterfly plot would no longer have bi-stable intersection points, indicating failure of the bit-cell to hold a required data state.

In the 6T cell, the read operation is performed by precharging the bit-lines (*BLC/BLT* in Figure 5-2(b)) and then asserting the word-line (WL) to turn on the access transistors (M5 and M6). The storage node which stores a "0", for instance *NT*, causes the bit-line

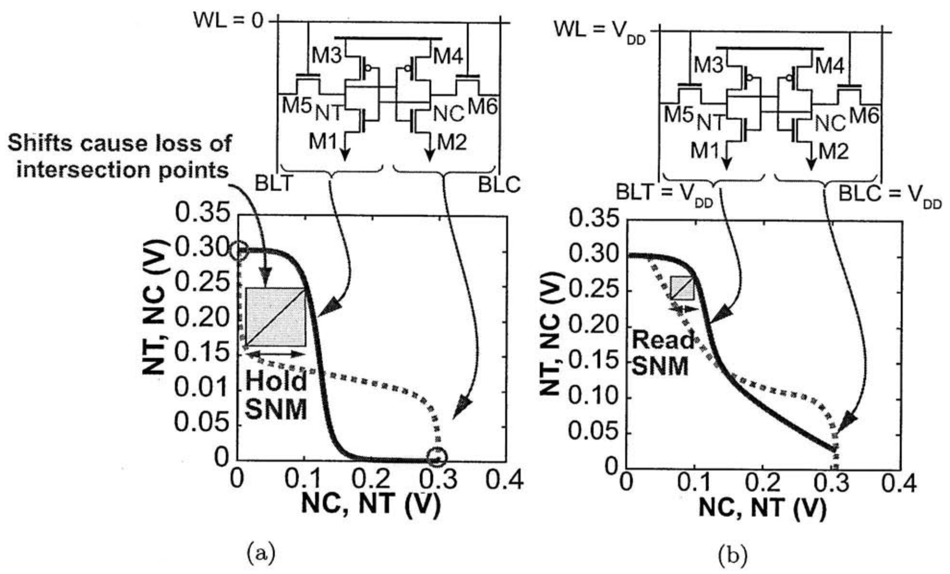


Figure 5-2: (a) Hold static noise margin in a conventional 6T bit-cell in 65nm. (b) Read static noise margin in the 6T cell. The word-line is asserted while transistor M1 must fight M5 to pull node *NT* low.

BLT to discharge. However, since the bit-line is initially precharged, M5 tends to pull *NT* high while the driver device M1 attempts to pull it low. The fight between M1 and M5 raises the voltage at NT. Accordingly, the butterfly plot for a 6T cell during read is squashed on one end as can be seen in Figure 5-2(b).

As V_{DD} is decreased, both read and hold SNM correspondingly become smaller. Moreover, as is apparent from Figure 5-2, the read SNM is considerably smaller than hold SNM and thus limits low-voltage operation [99, 112]. We confirm this for the target $0.13\mu m$ process technology through 50k-point Monte Carlo simulations of a 6T bit-cell. Figure 5-3 plots histograms of the hold and read SNM with local variation, at 0.5V, 85^oC (the worst case temperature corner) and nominal global process conditions. Local variation degrades the read SNM to nearly 0V, and in fact, the worst-case read SNM becomes negative once global variation is taken into account. This necessitates a different bit-cell topology as will be discussed in the next section.

Write Margin

Similar to the read operation, a successful write to a 6T bit-cell hinges on the relative strengths of two transistors – the access device must overcome the load device in order to write a new value into the bit-cell. One metric of a bit-cell's write margin is the trip voltage

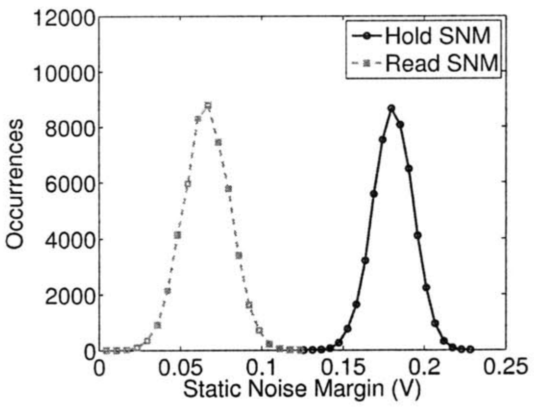


Figure 5-3: Histograms of the hold and read SNM of a 6T bit-cell at 0.5V, 85^oC, and under nominal global conditions. Data is obtained from 50k-point Monte Carlo simulation with local variation.

[113], and a bit-cell with a high trip voltage is easier to write into. Figure 5-4 shows a histogram of trip voltage of a 6-T bit-cell in the target $0.13\mu m$ technology at 0.5V, under worst-case global conditions and local variation. Here, a significant number of cells operate on the verge of write failure, which must be addressed especially in the design of large bit-cell arrays. However, improving the write margin through device sizing is again ineffective. Since reading requires the driver devices to be stronger than the access transistor, while writing dictates that the access transistor must overcome the load device, enforcing these conditions through device sizing would lead to impractically large bit-cells.

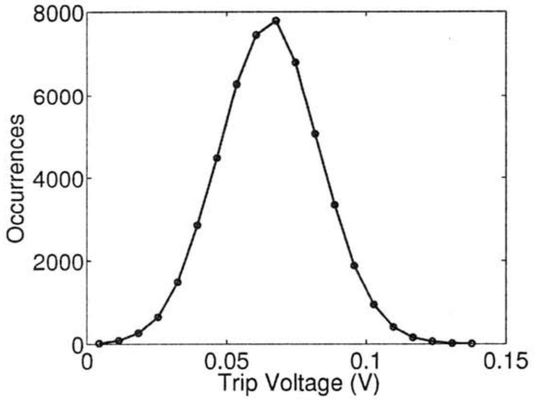


Figure 5-4: Histogram of the trip voltage with local variation, at 0.5V, -10^oC and weak-NMOS strong-PMOS global corner. Data is obtained from 50k-point Monte Carlo simulation with local variation.

Bitline Leakage

Thus far we have considered functionality of the 6T cell in isolation during reading, writing and data retention. However, to ensure correct sensing during a read, we must also consider the interaction between the accessed cell and its unaccessed neighbors. Figure 5-5 illustrates a typical sensing scheme in 6T SRAMs. The bit-lines are first precharged, then the read current in the accessed bit-cell discharges one of the bit-lines to develop a voltage differential. This is then amplified by a sensing circuit. At low voltages, the read current in the worst case bit-cell is much less than the mean, due to the exponential dependence of current on local V_t variation. The worst case cell thus severely limits the read speed at low voltages. Moreover, because the ratio of on to off currents (I_{on}/I_{off}) in a transistor is much reduced at low voltages, the read current on one bitline can be less than the aggregate leakage currents in the unaccessed bit-cells on the other bitline, causing read errors.

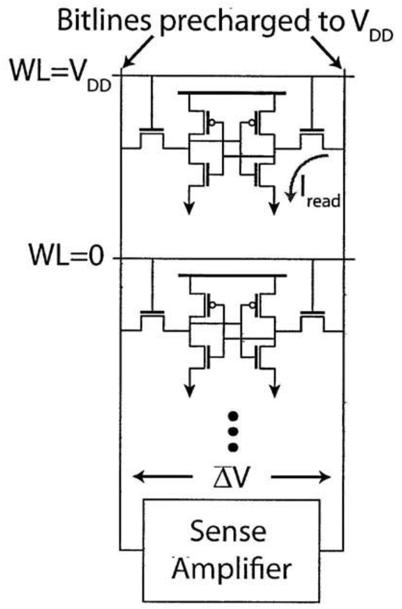


Figure 5-5: A typical sensing scheme in 6T SRAMs.

For the targeted $0.13\mu m$ process, Figure 5-6 shows the ratio of the worst case read current to the worst case aggregate bitline leakage current in a column of 256 bit-cells. At the strong process and temperature corner, the worst case bitline leakage exceeds the read current, making reliable sensing impossible. A method to decrease the bitline leakage is proposed in [105] and will be described in Section 5.2.4.

From the data presented above, it is clear that a conventional 6T SRAM in the target

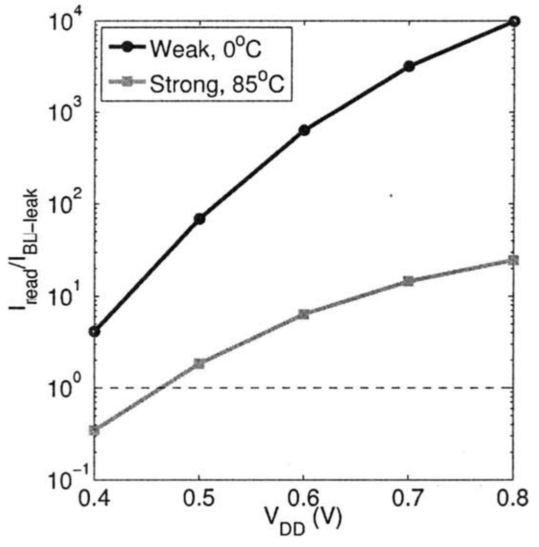


Figure 5-6: Ratio of the worst case read current to the worst case aggregate bitline leakage current in a column of 256 bit-cells.

$0.13\mu m$ technology suffers from degraded read and write margins as well as unreliable sensing at low voltages. The next section describes the design of a low voltage SRAM that addresses issues with read and write functionality.

5.2 Enabling Low Voltage Operation

5.2.1 Bit-Cell

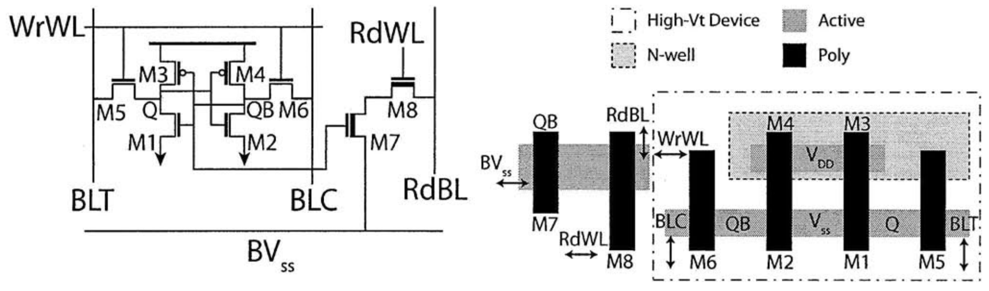
The SRAM in the processing platform employs an 8-transistor (8T) bit-cell which has been demonstrated in previous low-voltage SRAMs [112, 105, 109]. Pictured in Figure 5-7(a), the 8T bit-cell consists of the conventional 6T cell and a 2T read buffer. The read buffer serves to isolate the internal storage node from the bitline. Consequently, the internal node is not disturbed during a read, and the read SNM is the same as the hold SNM. In addition, the 8T topology allows separate optimization of the bit-cell for read and write functionality. In a 6T bit-cell, reading and writing introduce opposite sizing constraints on the access device as discussed previously. In the 8T bit-cell, the access device can be sized to aid writing and the read buffer sized to improve read speed.

The target $0.13\mu m$ process technology features low-leakage and standard transistors (high- and low-V_t devices respectively). This additional degree of freedom allows us to simultaneously improve speed and leakage over a single-V_t design. Since the processing platform employs low-V_t standard cells, the critical read path in the SRAM must also use low-V_t devices in order to achieve comparable speeds. However, the 6T storage cell does not lie in the read path, and can thus utilize high-V_t transistors to reduce the array leakage. Although the high-V_t access transistors would slow down the write operation, the speed decrease is mitigated by the write assist mechanism in this SRAM as detailed later. The leakage currents and read currents of bit-cells with various combinations of high- and low-V_t transistors are reported in Table 5.2. Using low-V_t devices in the read buffer improves read current by 3.5×, while using high-V_t devices in the storage cell reduces leakage by 2.92×, compared to a bit-cell with all low-V_t devices.

Table 5.2: Leakage and read currents of 8T bit-cell with different configurations of high- and low-V_t transistors. Simulations are performed at V_{DD}=1V, nominal process and temperature.

6T storage cell	2T read buffer	8T Cell Leakage Current	read current
High-V_t	High-V_t	8.29pA	$14.7\mu A$
High-V_t	Low-V_t	9.95pA	$51\mu A$
Low-V_t	Low-V_t	29.1pA	

The layout of the resulting bit-cell is shown in Figure 5-7(b), which satisfies special design rules concerning the spacing between high- and low-V_t transistors. The layout follows a "tall-cell" design where the word-lines are routed horizontally and the bitlines routed vertically. This layout was chosen so that BV_{ss} at source terminal of M7 can be routed horizontally across all other bit-cells in the same row, the reason for which will be discussed in Section 5.2.4.



(a) (b)

Figure 5-7: (a) 8-transistor, multi-V_t SRAM bit-cell. (b) Layout of the multi-V_t bit-cell.

5.2.2 Analyzing Write Assist Schemes

Since the write operation to an 8T bit-cell is the same as that of a 6T bit-cell, the challenge of reduced write margin outlined in Section 5.1 remains. A successful write requires that the access device (M5/M6) overpower the load device (M3/M4). Accordingly, techniques have been proposed to aid writing by strengthening the access device or weakening the load device, as summarized in Table 5.3. For example, [103] lengthened the access transistors to increase their drive current in sub-threshold (taking advantage of reverse short channel effects). Unfortunately, this weakens the access devices at nominal voltages and is thus unsuitable for a 0.5V-1V SRAM. In [101, 104], authors gate the power supply of the bit-cell during a write operation and let the V_{DD} rail float to a lower voltage, thereby weakening the PMOS load devices. The 8T SRAM in [105] takes this one step further and explicitly drives the V_{DD} rail to an intermediate voltage rather than simply allowing it to float. In addition, the design boosted the write word-line ($WrWL$) by 50mV to strengthen the access devices. These write assist schemes enable write functionality at low V_{DD} but introduce a power overhead at nominal V_{DD} when they are no longer needed. Accordingly, [109] proposes a reconfigurable write assist circuit which combines the approaches in [101] and [105] – depending on the operating voltage, the cell supply is either kept at full-rail, allowed to float, or actively driven to an intermediate voltage.

It should be noted that the above write assist schemes that involve reducing the bit-cell supply or boosting the word-line prevents column interleaving [106]. Two approaches have been proposed to address this limitation [108, 110]. Table 5.3 provides a summary of the above write assist techniques.

The SRAMs in [105, 109, 101] rely primarily on reducing the bit-cell supply voltage to enable writing at low voltages. Here, we analyze the energy consumption of two write assist techniques: reducing the cell supply and boosting the write word-line. These two techniques are not mutually exclusive and can be used together. However, if only one is needed to overcome variation in a particular design, it is useful to understand the energy trade-offs between the two approaches. We will refer to these as virtual V_{DD} and word-line boosting respectively in the subsequent discussion.

The virtual V_{DD} technique has several main sources of energy overhead. After reducing the cell supply (VV_{DD}) during a write cycle, the supply must be charged back up to V_{DD}.

Table 5.3: Proposed techniques to aid writing in low-voltage SRAMs.

Reference	Summary
Calhoun, ISSCC 2006 [101]	Floating bit-cell supply voltage
Zhai, ISSCC 2007 [104]	Floating bit-cell supply and ground voltage
Verma, ISSCC 2007 [105]	Drive bit-cell supply to intermediate value; boost word-line
Sinangil, ESSCIRC 2008 [109]	Reconfigurable between driving bit-cell supply, floating supply, and keeping supply constant
Chang, ISSCC 2008 [106]	10 bit-cell supporting column interleaving
Sinangil, A-SSCC 2009 [110]	New bit-cell array architecture enabling column interleaving

The capacitance on VV_{DD} includes the gate-source capacitance of the load devices, the wire capacitance on VV_{DD}, the N-well capacitance, and decoupling capacitance. Second, as pointed out by [99], a short circuit current path arises when VV_{DD} is pulled low, flowing between one of the bitline drivers and the VV_{DD} driver, as illustrated in Figure 5-8. Finally, designs with VV_{DD} employ a folded-row layout shown in Figure 5-9(a) [92] where each SRAM row is folded into two in the physical layout such that VV_{DD} can be shared across adjacent rows to achieve a dense layout. This doubles the wiring capacitance on the bitlines compared to a conventional layout.

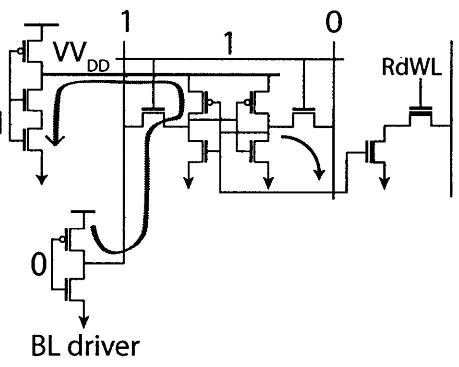
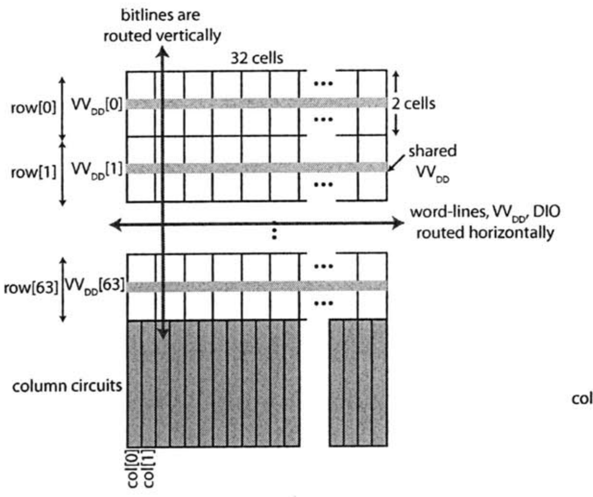


Figure 5-8: Short circuit current path between bitline driver and VV_{DD} driver when VV_{DD} is being pulled low.

In word-line boosting, the primary energy overhead comes from the need to temporarily boost the word-line voltage above V_{DD}. A boosting circuit was proposed in [105] for another



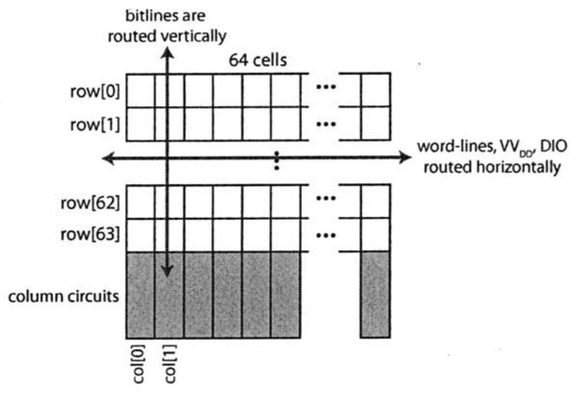

bitlines are routed vertically
64 cells
row[0]
row[1]
word-lines, VV_{DD}, DIO routed horizontally
row[62]
row[63]
column circuits
col[0]
col[1]
(b)

Figure 5-9: (a) Folded-row layout of a 64 row×64 column sub-block, enabling VV_{DD} to be shared across adjacent rows in the layout. (b) Conventional row layout of a 64×64 sub-block employed in the word-line boosting technique.

purpose but can be applied to word-line boosting. In the circuit shown in Figure 5-10, charge is stored on the capacitor C before the write word-line is to be asserted. During a write cycle, $WrWL$ is first charged to V_{DD}, then the charge stored on the capacitor is transferred onto $WrWL$, increasing its voltage above V_{DD}. The capacitor in the boosting circuit can be sizable since it must store sufficient charge to boost a large capacitance on $WrWL$. Further, the circuit timing should be well controlled: before *boostEn* rises and applies the boosting, $WrWL$ should be charged to nearly V_{DD} and transistor *MP1* should be turned off, otherwise the charge stored on C would be wasted. On the other hand, the bit-cells can be laid out in a conventional manner (Figure 5-9(b)), which results in shorter bitlines with approximately 2× lower wire capacitance.

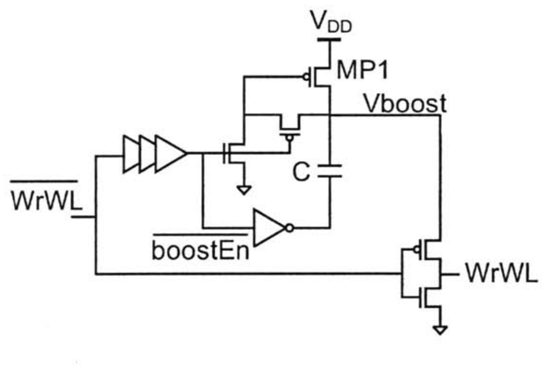


Figure 5-10: Circuit to boost $WrWL$ above V_{DD} to assist writing at low voltages [105].

Two SRAM sub-blocks each employing one of the above write assist techniques were simulated with extracted wiring parasitics. Figure 5-11 plots the energy breakdown at 0.5V of a write operation using the techniques. Due to its long and narrow layout, VV_{DD} consumes less energy in driving horizontally routed signals, as reflected by the smaller energy components in *WrWL*, array control, and other global signals. Conversely, VV_{DD} requires more energy to drive vertical bit-lines. Although word-line boosting requires driving a signal above V_{DD}, this analysis shows that the actual energy involved is only a small fraction of the total write energy. It is noteworthy that reducing and restoring VV_{DD} introduces considerable overhead and contributes to more than one-third of the total write energy.

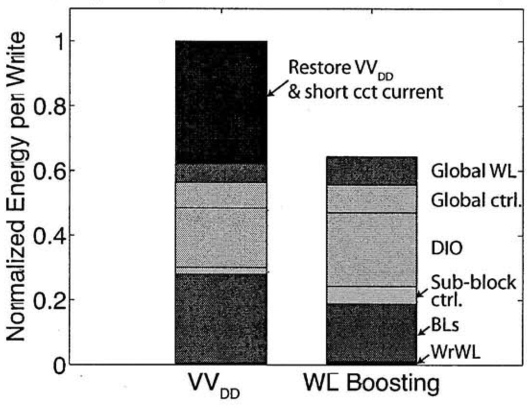


Figure 5-11: Write energy breakdown in two SRAM sub-blocks employing two write assist techinques (normalized to energy of VV_{DD} scheme). Simulated at 0.5V with wiring parasitics extracted from layout. Here, *DIO* refers to the data bus connecting 16 sub-blocks internal to the SRAM and used in both reading and writing.

Differences in row and column capacitances affect read energy as well. Figure 5-12 shows the energy breakdown during a read cycle. Again, VV_{DD} consumes more energy in precharging the read bit-lines, but less energy in driving global signals. Importantly, a large portion of the read energy is consumed when driving the read results onto the large *DIO* bus (the data bus connecting 16 sub-blocks internal to the SRAM and used in both reading and writing). The boosted word-line design has more capacitive loading on *DIO* because its shorter and wider layout requires *DIO* to span longer distances. These two opposing effects result in VV_{DD} consuming slightly less energy (1pJ) per read than boosted word-line.

Due to the trade-off between read and write, the energy of both techniques are similar when averaged over reads and writes. In this work we employ word-line boosting since it leads to a simpler SRAM layout; the VV_{DD} technique requires the column circuits to fit

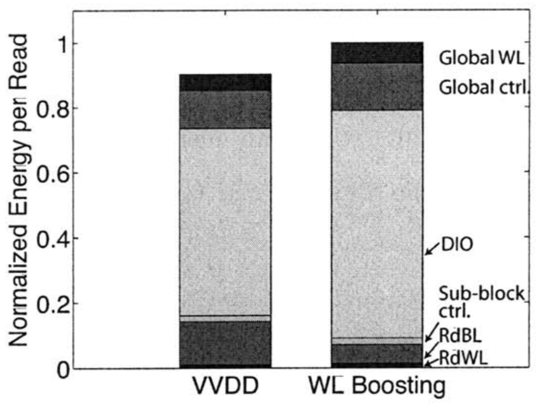


Figure 5-12: Read energy breakdown in the two SRAM sub-blocks, normalized to that of boosted word-line scheme. Simulated at 0.5V with wiring parasitics extracted from layout.

into half the bit-cell pitch.

It is possible to employ both techniques in a design. One reason for doing this would be if one technique alone does not enable reliable writing, as was the case in [105]. Another reason is to trade off the energy overhead of VV_{DD} at the cost of boosting the word-lines to a higher voltage. In either case, the need to share the VV_{DD} rail requires the use of the folded-row layout (Figure 5-9(a)). This implies that most components of the sub-block energy (*Global WL, global control, DIO, sub-block control, BL*) would be the same as in a VV_{DD}-only design. As seen from Figure 5-11, a design combining the two techniques must reduce the energy overhead of VV_{DD} by 77% in order to achieve write energy comparable to a design with only boosted word-line.

The preceding analysis revealed that the *DIO* energy is a major component in both reading and writing. This will be addressed further in Section 5.3.

5.2.3 Read Current Modeling

As mentioned in Section 5.1, the read current (I_{read}) in an SRAM bit-cell discharges the bitline to develop a voltage differential for sensing. This bitline discharge time is a large component of the read access delay of an SRAM. Consequently, accurately predicting the smallest read current in the bit-cell array is key to predicting the speed of an SRAM.

The read current for velocity-saturated devices in above-threshold is linearly related to V_t, and as a result, it is common practice to model the read current by a Gaussian distribution as shown in Figure 5-13. Based on the distribution, one can extrapolate the expected worst case (smallest) read current in a bit-cell array of a given size. For example, in the design of large bit-cell arrays, designers are interested in the "5σ" read current that is 5 standard deviations smaller than the mean read current. A similar analysis can be performed in deep sub-threshold, when the read current is lognormally distributed. Unfortunately, this method breaks down when the supply voltage is near threshold, which is the targeted lowest V_{DD} of the biomedical platform and other low-voltage systems (e.g. [28, 114]). In this region, the observed data cannot be fitted accurately to a normal or lognormal distribution. To find the worst case read current, one would need a large number of simulations which grows proportionally with the array size, or settle for an inaccurate estimate based on one of the above distributions.

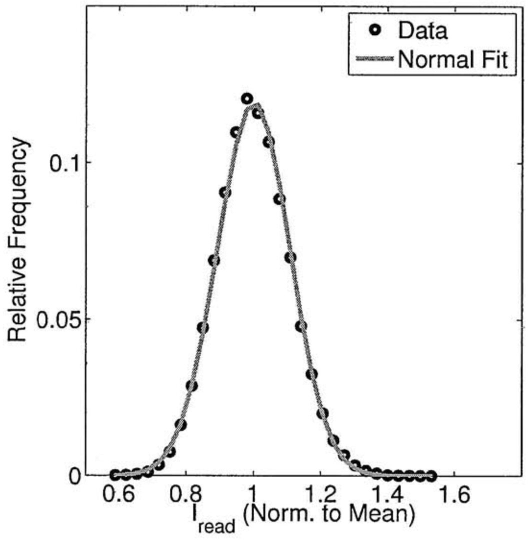


Figure 5-13: Read current in above-threshold can be modeled by a Gaussian distribution.

The difficulties in modeling can be attributed to two major factors. First, when V_{GS} (and V_{DD}) is close to V_t, the transistor is in moderate inversion and its drain current deviates from behavior predicted by both strong- and weak-inversion models[1] [90]. Further, when V_{DD} is close to the nominal V_t, local V_t variation, by increasing or decreasing V_t, can then put a device in the above- or sub-threshold mode of operation. In light of the latter observation, we propose modeling the read current distribution at $V_{DD} \approx V_t$ as the weighted sum of a Gaussian and a lognormal distribution. The distribution shown in Equation 5.1 has six parameters: α_1 and α_2, the weights, μ_1 and σ_1, the two parameters of the normal

[1]We use the terms strong- and weak-inversion interchangeably with above- and sub-threshold.

distribution, and μ_2 and σ_2, the two parameters of the lognormal distribution.

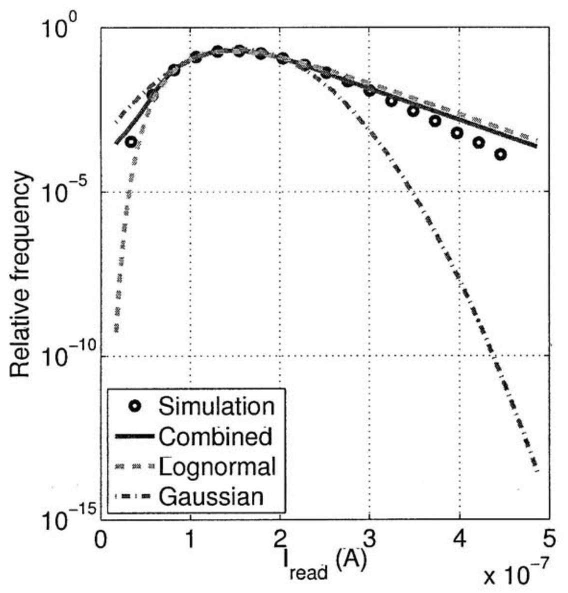
$$f(x) = \frac{\alpha_1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} + \frac{\alpha_2}{x\sqrt{2\pi\sigma_2^2}} e^{-\frac{(\ln x-\mu_2)^2}{2\sigma_2^2}} \tag{5.1}$$

While it is straightforward to fit a dataset to either a Gaussian or a lognormal distribution, since the maximum likelihood estimates of the parameters are given by closed-form expressions, the same is not true for the weighted sum in Equation 5.1. One can consider using the EM algorithm [115] to estimate the parameters given a dataset. Alternatively, for ease of implementation, we opted to use the curve fitting capabilities of MATLAB to fit a histogram of I_{read} to the model in Equation 5.1. To help the curve fitting function converge, we must specify reasonable initial guesses for the parameters. The following steps enabled convergence in our experiments.

1. Fit I_{read} data to Gaussian distribution only and record the two parameters that result.
2. Fit I_{read} data to lognormal distribution only and record the two parameters that result.
3. Use the parameters from steps 1 and 2 as initial guesses for μ_1, σ_1, μ_2, σ_2. Provide a reasonable guess for α_1 and α_2.
4. Obtain histogram of the I_{read} data.
5. Run the curve fitting function with initial guesses to fit Equation 5.1 to the histogram.

Using the above approach, we fit the above model to simulated I_{read} in a 1 million-point Monte Carlo simulation at V_{DD}=0.5V and 0.6V. The circle markers in Figure 5-14(a) show the simulated histogram of I_{read} at 0.5V. Recall that in SRAM design, we are concerned with the left tail in finding the smallest I_{read} in a large SRAM array. Comparing the dashed lines to the simulated data towards the left of the graph, we see that a Gaussian distribution underestimates the weak-cell I_{read}, while a lognormal distribution overestimates it. The solid line representing the model in Equation 5.1 provides a better fit to the simulated data. A similar discussion applies to the data taken at V_{DD}=0.6V, reported in Figure 5-14(b).

For near-threshold supply voltages, the proposed model can be used to predict the smallest read current in a bit-cell array, which is used in sizing the read buffer in the bit-cell as follows:



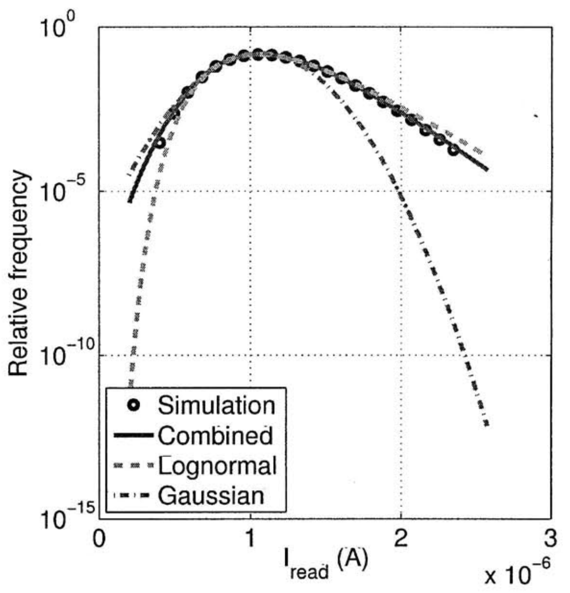


Figure 5-14: Simulated read current distribution from 1 million-point Monte Carlo simulation. Dashed lines represent fitted Gaussian and lognormal models, and solid line indicates the proposed model in Equation 5.1. (a) and (b) plot data at V_{DD}=0.5V and 0.6V respectively.

1. Integrate the model in order to obtain the cumulative distribution function (CDF) of the read current. In MATLAB, the integral is approximated numerically by summing over the y-axis value of the model at each I_{read} multiplied by the step in I_{read} (see Figure 5-14). [2]
2. From the CDF, find the value of I_{read} corresponding to *1/array size.* This is the expected smallest read current ($I_{read-weak}$) in the array.
3. Compute the bitline discharge time as $\frac{C_{BL}V_{DD}}{I_{read-weak}}$, where C_{BL} is the bitline capacitance.
4. Add the bitline discharge time to the propagation delay of other components of the read path to find the read access time of the SRAM.

5.2.4 Assisting Read Operation

Although the 8T bit-cell removes the read SNM limitation, it does not address the issue of bitline leakage. In the 8T cell, bitline leakage imposes a similar problem to that of a 6T cell described in Section 5.1. The 8T cell has one read port and hence relies on pseudo-differential sensing – after a period of time allocated to bitline discharge, read bitline

[2] A scaling factor should be applied in order for the CDF to sum to 1.

voltage (RdBL) is compared to a fixed reference (VREF). In this situation, the critical case to consider is when the RdBL should remain precharged at V_{DD}, but is instead partially discharged by the leakage in the unaccessed cells.

Some degradation of the RdBL can be tolerated by the sensing circuitry, but the RdBL must not droop below VREF plus the offset of the sense amplifier. Based on statistical simulations of the bitline leakage and sense amplifier offset, the droop on RdBL is acceptable at the weak process corner but is too large at the strong corner, reflecting the findings in Figure 5-6. To reduce bitline leakage, we use the approach proposed in [97] and illustrated in Figure 5-15. The source node of M7 in the read buffer is driven to V_{DD} on the unaccessed bit-cell, which eliminates the source-drain sub-threshold leakage current in the read buffer.

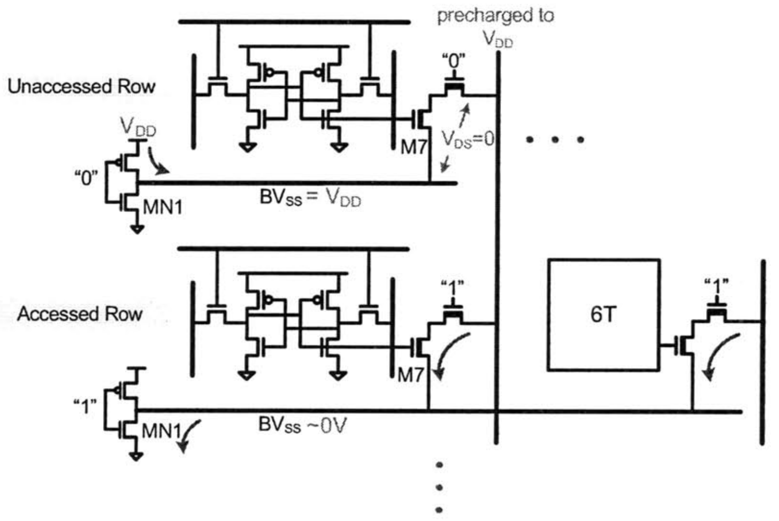


Figure 5-15: Approach to eliminate source-drain sub-threshold leakage current on read bitline [97].

For the SRAM row being accessed, the NMOS MN1 (Figure 5-15) in the read buffer driver must sink the read currents of all cells on the row. In the unaccessed rows, however, MN1 should be small to limit leakage. To sidestep this sizing trade-off, a charge pump from [97] can be used to boost the gate of MN1 in the accessed row, dramatically increasing its drive current at low voltages. Although [97] adds a dedicated boosting circuit for the read assist, in this work we can reuse the circuit for boosting the write word-line as shown in Figure 5-16 since reading and writing are mutually exclusive. Further, the capacitor C in this circuit is sized for boosting a large write word-line capacitance by 100mV, and can therefore provide a large boost to the small gate capacitance of the read buffer driver.

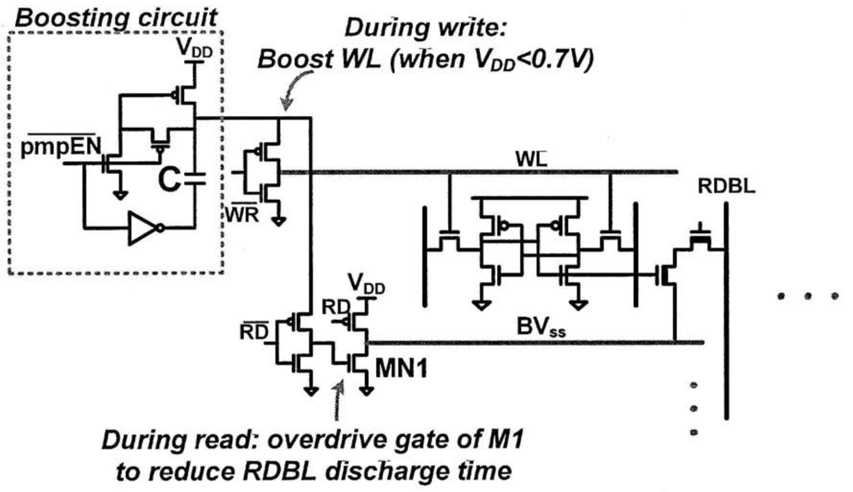

Boosting circuit
V_DD
pmpEN
C
During write:
Boost WL (when V_DD<0.7V)
WR
WL
RDBL
V_DD
RD
RD
MN1
BV_ss
During read: overdrive gate of M1
to reduce RDBL discharge time

Figure 5-16: Circuit shared between boosting word-line during write and boosting gate of MN1 during read.

5.3 Glitch Energy Reduction

Figure 5-17 shows the top-level organization of a 64kb SRAM macro. 16 bit-cell sub-blocks of 64 rows by 64 columns are connected via a tri-state data bus (*DIO*) used for both reading and writing. During a read operation, the accessed sub-block drives the value being read onto the *DIO* bus to the top-level output buffers. During a write, the global control block drives the input data via *DIO* to the bitline drivers in the sub-block. In low-voltage SRAMs, the energy to drive the *DIO* bus can be particularly large for two reasons. First, full-swing signaling is employed for increased robustness instead of the low-swing schemes found in above-threshold SRAMs. Second, the area density of low-voltage SRAMs tend to be lower because having fewer bit-cells in a column helps improve read speed and relieve bitline leakage. However, *DIO* must now span a larger physical area for the same memory capacity. To lower the *DIO* bus energy for low-voltage SRAMs, we propose a self-timed technique to reduce glitch energy.

A detailed diagram of the SRAM column circuits is shown in Figure 5-18. In a read operation, the $RDBL$ is allowed to discharge for a set period of time. Subsequently the sense amplifier (implemented as a StrongArm latch) is strobed, and the result stored in an SR latch. The latched result is then driven onto the *DIO* bus by a tri-state buffer. Since the sense amplifier output is latched, the *DIO* bus does not change until the next read cycle, providing the the next read occurs on the same sub-block. On the other hand, if the next

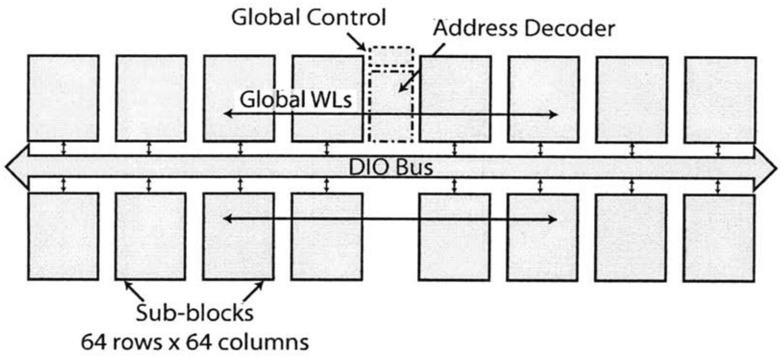


Figure 5-17: Top-level organization of a 64kb SRAM macro.

read occurs on another sub-block, then the new sub-block would drive data previously held in the SR latches before the newly accessed data arrives, thus causing glitches on *DIO*. This is illustrated in Figure 5-19. From a software point of view, consecutive memory reads to different sub-blocks occur frequently, for example when the program instructions are stored in one block of memory while the data is stored in another, or when the program is accessing the software stack.

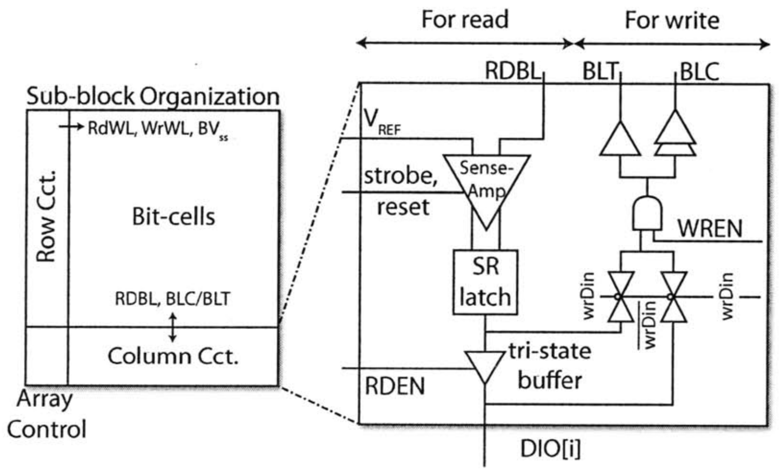


Figure 5-18: Detail of SRAM column circuits.

One method to prevent glitches is to enable the tri-state drivers only after each sense-amplifier has resolved. At the beginning of each read cycle, the differential sense-amplifier outputs are both reset to V_{DD}, then one output is discharged to ground during evaluation. Accordingly, an XOR of the sense-amplifier outputs can be used to indicate that evaluation has completed. The straightforward way to eliminate glitches is then to gate the tri-state

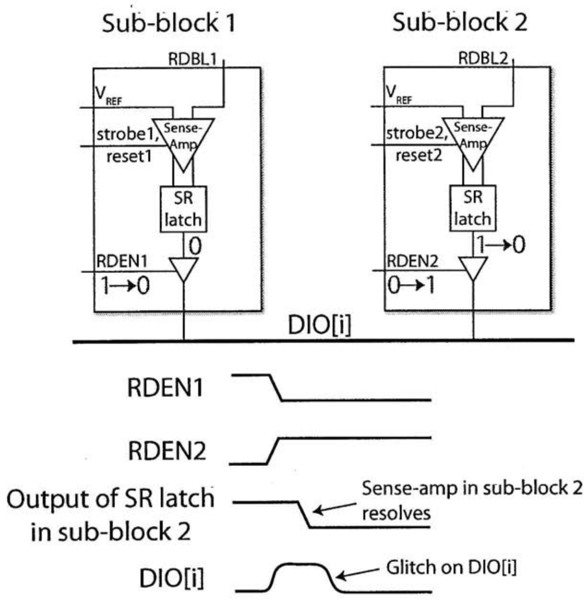


Figure 5-19: Example of glitch occurring on *DIO*.

buffers with an XOR of the sense-amplifier outputs, as shown in Figure 5-20. If, on average, half of the *DIO* lines glitch when switching between sub-blocks, then gating the tri-state buffers reduces the switched capacitance by 28pF, a substantial amount.

Upon closer examination, however, we see that adding an XOR plus an AND gate to every column imposes considerable leakage overhead. Whether this addition provides net energy savings depends on the trade-off between glitching energy reduction and leakage energy increase. This is analogous to the trade-off in power gating, and therefore break-even time analysis can be applied. Here, the break-even time is $t_{be} = E_{glitch}/P_{leak-logic}$, or the ratio of the glitching energy reduction to the additional leakage power. Table 5.4 lists the three quantities at V_{DD}=1V and 0.5V.

Table 5.4: The break-even time of adding logic to gate tri-state drivers for reducing glitches on *DIO*. The added logic must be activated with the break-even time in order to provide net energy savings. Leakage power overhead refers to the leakage of the two gates in the shaded box of Figure 5-20, which are added to every column in the SRAM.

V_{DD}	Glitching energy savings	Leakage power overhead	Break-even time	Equivalent number of clock cycles
0.5V	7.03pJ	79.4nW	88.6μs	8.9
1.0V	28.1pJ	150nW	188μs	1880

To achieve net energy savings, there must be consecutive SRAM reads to two different

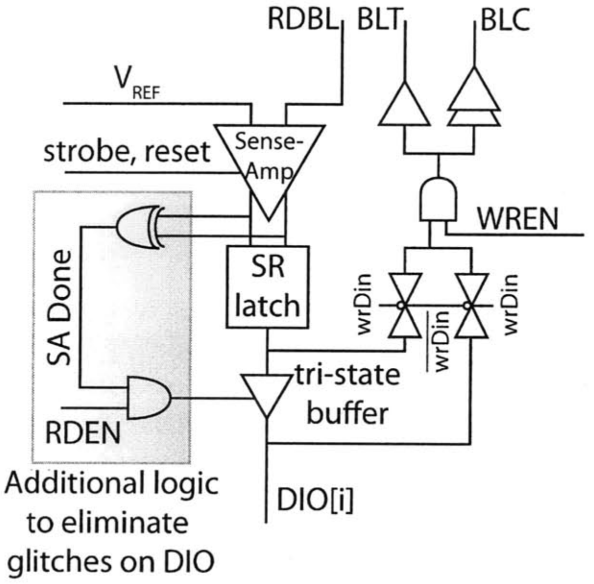


Figure 5-20: SRAM column circuits with additional logic to enable tri-state buffer only after the sense-amplifier resolves.

sub-blocks within t_{be}. At 1V this is a likely occurrence since t_{be} equals 1880 clock cycles. Conversely, at 0.5V t_{be} translates into only several clock cycles, and the added logic may well consume more energy than it saves.

As is apparent from the discussion of SRAM challenges in Section 5.1, in SRAM design it is common to account for the worst case scenario, which in this context implies addressing all glitches on *DIO*. Since the evaluation times of all 64 sense-amplifiers in the sub-block differ due to local process variation, each must have its own gating logic to completely eliminate glitches. However, we can instead consider the average case. On average, one sense-amplifier selected at random will be slower than half of the sense-amplifiers in the sub-block. By using one sense-amplifier to gate all tri-state buffers in the sub-block, we can expect to reduce glitches on half of the *DIO* bus.

We can further extend this to selecting two sense-amplifiers. In this case, the slower of the two selected amplifiers determines how many tri-state buffers can be effectively gated to remove glitches. The expected savings can be computed as follows. Given the integers 1 to 64 (corresponding to 64 sense-amplifiers ordered according to their delays), randomly draw two integers without replacement (X, Y), and find the expected value of their maximum.

Let $M = max(X, Y)$, then

$$E(M) = \sum M \times PMF(M)$$
$$= \sum_{M=1}^{64} \frac{M(M-1)}{{}_{64}C_2}$$
$$= 43.3$$

In other words, by selecting two sense-amplifiers to gate all tri-state buffers, on average we can remove glitches in 43 out of 64 bits on the *DIO* bus. This can be achieved at the relatively low cost of two XOR and two AND gates per sub-block, as illustrated in Figure 5-21. Table 5.5 reports the break-even times of using one and two sense-amplifiers to generate the gating signal. Using a small subset of the sense-amplifiers, we can achieve $\approx 2/3$ of the ideal glitching energy reduction while significantly reducing the leakage cost.

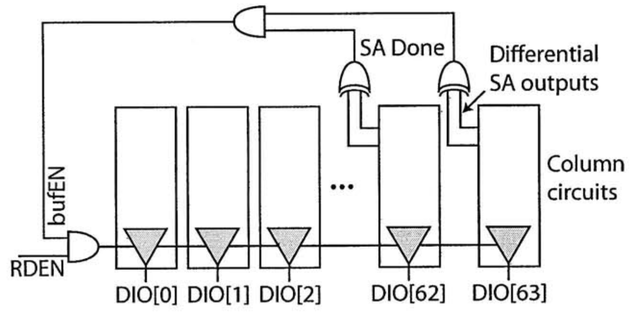


Figure 5-21: Generating tri-state enable signal (*bufEN*) for all 64 columns with the differential sense-amplifier outputs of two columns.

Table 5.5: The break-even time when selecting only one or two sense-amplifiers to generate gating signal for tri-state drivers.

V_{DD}	Glitching energy savings	Leakage power overhead	Break-even time	Equivalent number of clock cycles
Select one sense-amplifier				
0.5	3.52pJ	1.28nW	2.75ms	275
1.0	14.1pJ	2.42nW	5.82ms	58200
Select two sense-amplifiers				
0.5	4.69pJ	2.56nW	1.83ms	183
1.0	18.8pJ	4.84nW	3.88ms	38800

5.4 Measurements and Characterization

The SRAM, whose layout is shown in Figure 5-22, was integrated into the processing platform and fabricated in the target $0.13\mu m$ process. Since the SRAM is tightly integrated with the logic, it is not possible to measure its speed independently. However, measurements confirm that the system as a whole achieved the required performance in silicon. Figure 5-23(a) plots the frequency versus V_{DD} of the SRAM integrated with the logic. Since the SRAM has separate power pins from the logic, it is possible to measure the SRAM energy independently. Measurements (averaged over reads and writes) at room temperature are shown in Figure 5-23(b). The minimum energy point of the SRAM occurs at 0.6V. At 0.5V, the leakage energy component causes the total energy to increase.

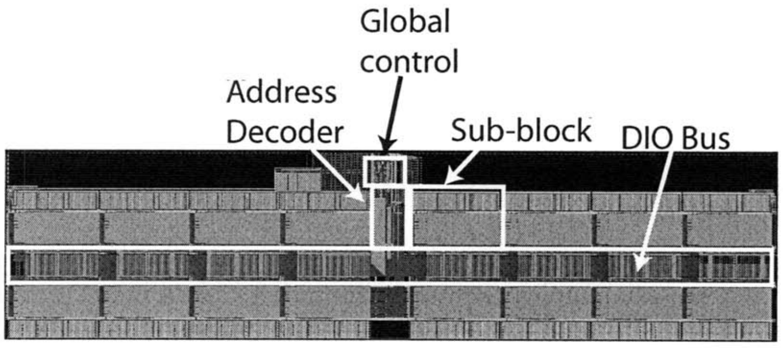
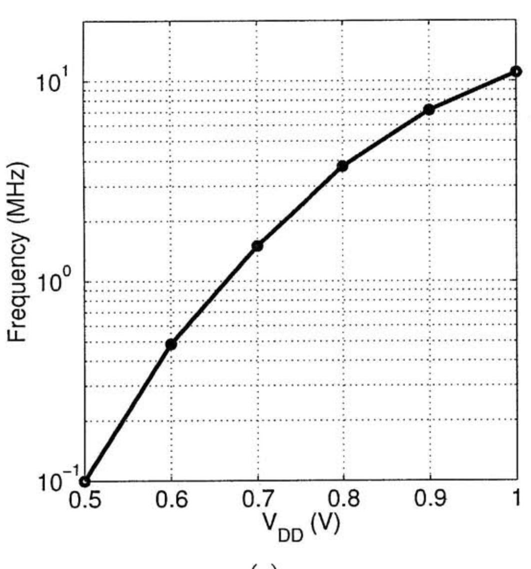


Figure 5-22: Annotated layout of 64kb SRAM macro in a $0.13\mu m$ process.

The SRAM was laid out with different components supplied by separate power pins such that we are able to measure the energy of individual components on silicon. Figure 5-24 shows the components of total energy at 0.6V and 1V.[3] At 1V, the energy consumed for driving global control signals (for example the global word-lines and sense amplifier strobe signal) forms the largest component of the total energy. The column circuits, which include the data bus drivers, are the second largest contributor. At 0.6V, the column circuits overtake global control as the largest energy component. Since the SRAM contains many copies of the column circuit (one for each column), the leakage power in the unaccessed columns becomes significant when integrated over a long clock period.

Table 5.6 lists the measured energy of two previously published 8T SRAMs with architectures similar to this work. The energy per access shown is averaged over reads and

[3]Note that the partitioning in silicon is somewhat different from the simulated energy breakdown.



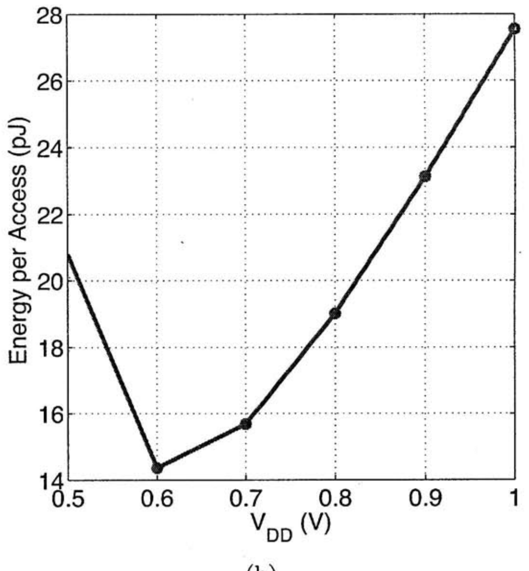


Figure 5-23: (a) Frequency versus V_{DD} of SRAM integrated with logic. (b) Energy per access versus V_{DD} of 64kb SRAM macro. Both measurements are taken at room temperature.

Table 5.6: Energy per access comparison with previously published SRAMs with similar architectures to this work.

Reference	Technology	Sub-Block Size	Minimum Energy Point	Minimum Energy/Access
Sinangil, ESSCIRC 2008 [109]	65nm	64×128	0.4V	11pJ
Kwong, ISSCC 2008 [26]	65nm	64×64	0.5V	14.5pJ
This work	$0.13\mu m$	64×64	0.6V	14.4pJ

writes. The SRAM in this work achieves an energy per access comparable to previously published SRAMs that were implemented in more advanced technologies.

5.5 Conclusions

An examination of the key functional metrics of an SRAM – the operational margin for reading, writing, and holding data – reveals that a 6-transistor SRAM in the target $0.13\mu m$ process cannot function reliably at 0.5V. In particular, the read and write margins, as well as substantial bitline leakage at the fast global process corner, must be addressed.

The SRAM design in the processing platform utilizes an 8-transistor bit-cell that is amenable to low-voltage operation. While low-voltage 8T SRAMs have been demonstrated

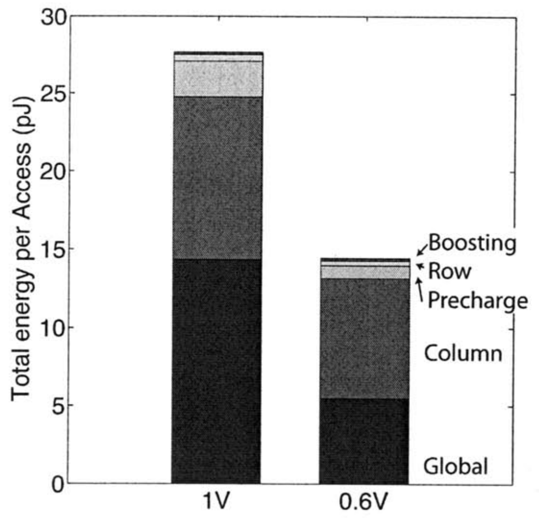


Figure 5-24: Energy breakdown of the 64kb SRAM macro at V_{DD}=1V and 0.6V (the minimum energy point). The energy is averaged over read and write accesses.

previously [112, 105, 109], this chapter analyzes two aspects that have not been addressed in prior work. We examine the energy of two common write assist schemes, reducing the bit-cell V_{DD} and boosting the word-line. It is shown that the first scheme requires considerably more energy for writing due to the overhead of restoring the bit-cell V_{DD} after a write access. The read energy of the first scheme is slightly lower as its tall physical layout results in a shorter data bus connecting the SRAM blocks. Next, we propose a model for the worst case read current in an SRAM array, intended for situations where the supply voltage is near-threshold ($\approx$0.5V) and conventional models become inaccurate.

The energy analysis above showed that a sizable portion is needed to drive a large data bus connecting blocks within an SRAM. Accordingly, we employ a self-timed scheme to reduce glitches on the data bus. However, we show that it is important to weigh the leakage overhead against active energy savings, particularly in SRAM design when a small circuit change is replicated many times across the array. The usual practice of anticipating the worst case and eliminating all glitches imposes excessive leakage overhead. Instead, we implement a scheme that considers the average case and remove most glitches at a small fraction of the leakage cost.

Chapter 6

Chip Measurements and Biomedical Application Demonstration

Having described the design and optimization of components in the biomedical platform, we now turn to measurement results of the prototype test-chip. The top-level block diagram of the test-chip is repeated in Figure 6-1 for convenience. Below is a list of modules on this platform and a brief description of their function.

- **Microcontroller (*μC*) Core**: a 16-bit RISC CPU supporting the MSP430 instruction set along with logic for software debugging (e.g. stepping through code, breakpoints).
- **Direct memory access (DMA)**: for efficiently copying data between any two memory-mapped locations in the main memory and peripherals.
- **JTAG**: a 4-wire interface for loading instructions and data into the main memory at start-up.
- **Timers and real time clock**: the timer generates timing intervals and pulse width modulated signals, while the real time clock is helpful for timestamping data.
- **Serial ports**: for communicating with external components such as the ADC and radio. Support UART, SPI, and I^2C protocols.
- **Multiplier**: a 32-bit hardware multiplier.
- **ADC interface**: for communicating with a custom low power ADC being developed

at MIT.

- **General-purpose I/O ports (GPIO)**: 40 pins that can function as digital input or output ports to interface to external components.
- **Power Management Unit (PMU)**: manages power gating of all other modules in the system, as discussed in Section 3.3.
- **FFT, CORDIC, FIR, Median**: custom hardware accelerators detailed in Chapter 2.
- **SRAM**: the main memory of the system which stores program instructions and data. The memory is custom-designed to operate across 0.5V-1V, as discussed in Chapter 5.

A die micrograph of the test-chip, implemented in a $0.13\mu m$ low power process, is shown in Figure 6-2 along with annotations of the various components. Since non-volatile memory was not available for this test-chip, the software program and data are loaded at power-up into the chip's SRAM through the JTAG interface. To facilitate testing and debugging, the memory address and data buses (*MAB, MDB*) and key internal signals are multiplexed to the general-purpose I/O ports so that we can observe them externally. Since most of the activity in the system appears on *MAB, MDB*, we can compare transactions observed on-chip to Verilog simulations to validate silicon functionality. Operation of the timers and serial ports were verified in preparation for a future system prototype integrating this digital processor with analog front-end and radio. Lastly, the accelerators can be verified by printing their outputs to the GPIO ports and checking the results against expected values.

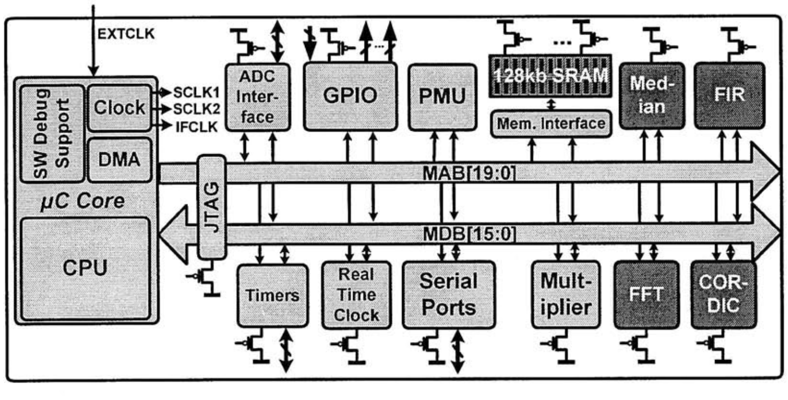


Figure 6-1: Block diagram of medical processor.

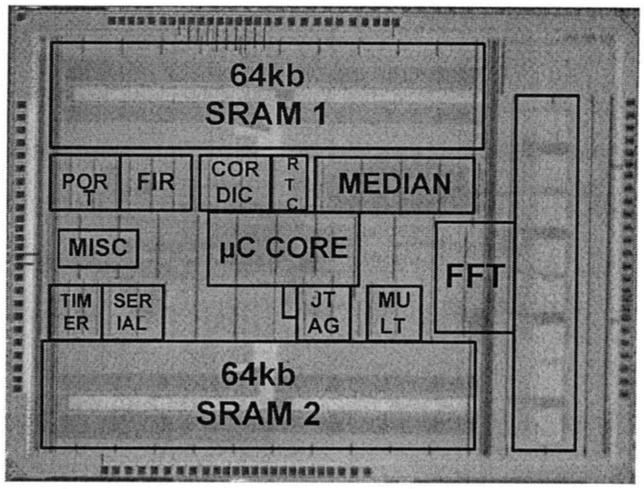


Figure 6-2: Die micrograph of biomedical signal processing platform fabricated in a low power $0.13\mu m$ process.

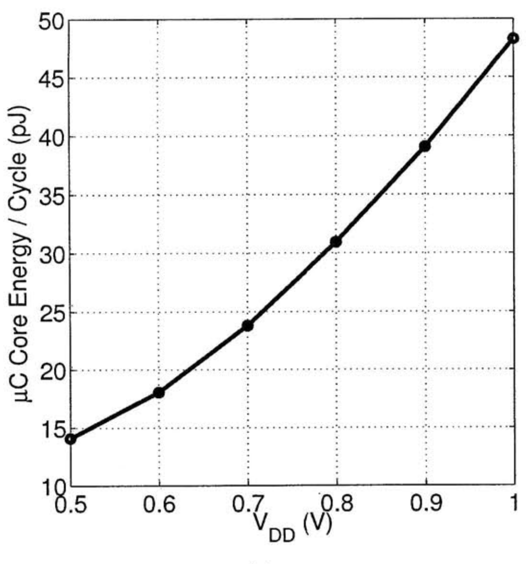
6.1 Top-Level Chip Measurements

The measured energy per clock cycle versus V_{DD} of the microcontroller core, which includes the 16-bit CPU, DMA, and software debug support logic, (μC core) is plotted in Figure 6-3(a). The core energy decreases monotonically with V_{DD}, reflecting its relatively high active energy component relative to leakage as explained in Chapter 4. For a 64kb SRAM macro, Figure 6-3(b) shows that its energy per access (averaged over reading and writing) reaches a minimum at V_{DD}=0.6V and increases at lower V_{DD}, which is consistent with its low activity factor.

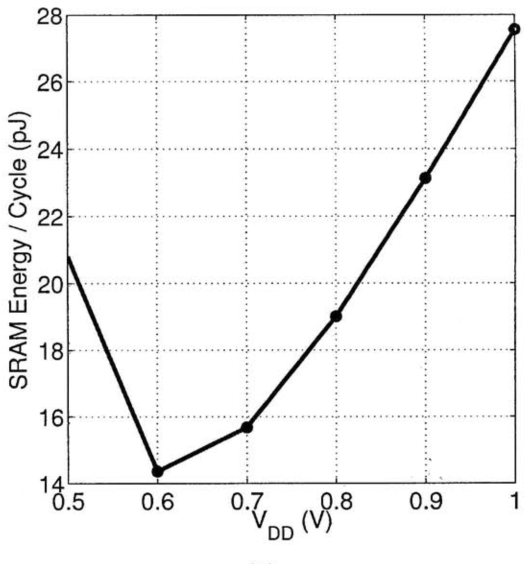
All components in the platform are designed to operate at the same frequency. The measured frequency across V_{DD} of the system is shown in Figure 6-4. The measured frequency lies within the range predicted by simulations.

6.1.1 Accelerator Measurements

This section details the system energy required to complete signal processing operations including the energy of transferring data to and from the accelerators. We compare the number of clock cycles and total energy to execute tasks in two ways. First, an operation (e.g. an FFT) is specified in C software, compiled with default compiler optimizations, then executed on the CPU and hardware multiplier. The MSP430 CPU does not include an integrated multiplier, and hence the compiler maps multiplications to a 32-bit hardware



(a)



(b)

Figure 6-3: (a) Energy per clock cycle versus V_{DD} of microcontroller core. (b) Energy per access (averaged over reading and writing) of a 64kb SRAM macro.

multiplier peripheral. Second, the same operation is computed with a hardware accelerator. Table 6.1 summarizes the number of clock cycles required in the two implementations, while Table 6.2 reports the total energy. Accelerators provide between 133 to 215× energy reduction for the listed operations. Of course, a complete application involves much more than an FIR filter or an FFT. We will quantify the energy savings provided by accelerators in the context of complete applications in Section 6.3.

Table 6.1: Number of clock cycles needed to execute signal processing tasks with 1) CPU and multplier, and 2) hardware accelerator.

Operation	Number of Clock Cycles	
	CPU & Multiplier	Accelerator
32-tap FIR Filter	1890	32
512-point Complex-Valued FFT	918880	6431
$\sin(x)$ (CORDIC)	3395	36
65-point Median Filter	1210	15

6.2 Accelerator Programming Model

While sophisticated C compilers exist for the MSP430 microcontroller core in this platform as well as for other microcontroller architectures (e.g. 8051, ARM), compiler support for

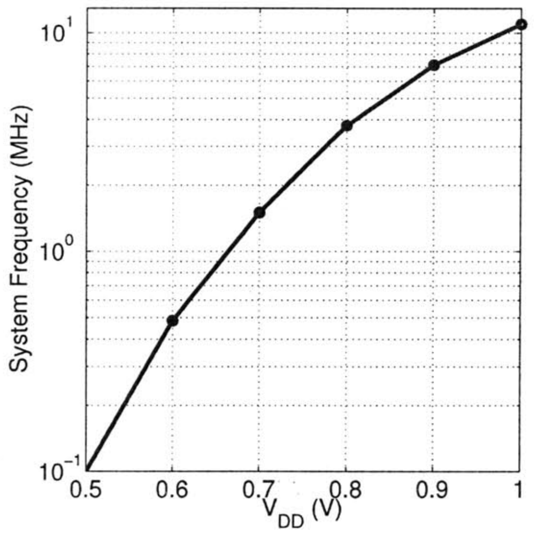


Figure 6-4: Frequency versus V_{DD} of all components in the platform.

Table 6.2: Energy per clock cycle and total energy to execute tasks with 1) CPU and multplier, and 2) hardware accelerator.

Operation	CPU & Multiplier		Accelerator		Reduction from accelerators
	Energy Per Cycle (pJ)	Total Energy (nJ)	Energy Per Cycle (pJ)	Total Energy (nJ)	
32-tap FIR Filter	93.2	176	38.1	1.22	144.4×
512-pt CVFFT	89.4	82148	95.8	616	133.3×
sin(x) (CORDIC)	82.3	279	36.1	1.30	215.2×
65-pt Median Filter	94.2	114	52.4	0.79	144.9×

accelerators is less well understood. Ideally, a compiler would be able to infer, from a high-level software description, portions of an application that can be mapped onto hardware accelerators. The design of such a compiler lies outside the scope of this thesis, but here we will address how the accelerators can be incorporated into a software application.

6.2.1 Wrapper Functions

Usage of the accelerators is very similar to the use of peripherals in the MSP430 and other architectures. A general sequence of events is as follows:

1. Program control registers to select desired features and mode of operation.
2. Provide the input data to be processed and start the accelerator.
3. Wait for the accelerator to complete.

4. Retrieve results.

There are several ways to wait for the accelerator to complete. As mentioned in Chapter 2, the accelerators can raise a DMA trigger or an interrupt at task completion. However, we have shown that interrupt handling can impose significant energy overhead. Alternatively, if the computation time is short, it may be more efficient for an application to poll the accelerator until it completes, thus avoiding the overhead of interrupt handling. Finally, since the latency of the accelerators is deterministic, an application can simply proceed to other tasks and fetch results from the accelerators after sufficient time has elapsed. In all cases except polling, the rest of the system can be clock- or power-gated while waiting for the accelerator.

As an example, the C software below illustrates the steps to initialize the FIR filter. When the filter completes one operation, the FIR interrupt service routine is launched and the result saved into a variable.

```
/************ Initialization Routine ************/
void init_FIR() {
    FIREN = 0;                  //Reset FIR by clearing FIREN bit
    FIRCTL0 = DECFILTSIZE-1;    //set FIR order
    FIRCTL1 = 0;
    FIRINTR = 1;                //Enable interrupt on completion
    FIREN = 1;                  //Start FIR accelerator
    //initialize FIR filter with data
    for(i=0; i<MAXSTORE; i++) {
        FIRWRDATA = eegSamp[chnl][i];
    }
    //now initialize with coefficients
    for(i=0; i<MAXSTORE; i++) {
        FIRWRCOEFF = DownSample[i];
    }
    return;
}


/************ Interrupt Service Routine ************/
#pragma vector=FIR_VECTOR
__interrupt void FIR_ISR(void) {
    firIntrVec = FIRIV;       //Clear the interrupt
    firOut = FIRRES;    //Get FIR output
    ...
}
```

In the above example, the programmer must be familiar with the structure of the FIR control registers and the order in which the data and coefficients should be provided. This is similar to the level of knowledge required in conventional embedded programming. However, we can also simplify the programmer's task by abstracting some details into wrapper functions. In the example shown below, the programmer can call a wrapper function *init_FIR* with arguments specifying the filter order, symmetry, and pointers to data. The wrapper function then programs the accelerator accordingly.

```
void main() {
    int firIn[32];
    int firCoeff[32];
    ...
    //call wrapper function
    init_FIR(31, 1, 1, firIn, firCoeff);
    ...
}
/************ A wrapper function provided to the programmer ************/
void init_FIR(int order, int symmetry, int intrEn, int* data, int* coeff) {
    FIREN = 0;                          //Reset FIR by clearing FIREN bit
    FIRCTL0 = order;                    //set FIR order
    FIRCTL1 = (symmetry) ? 0x2000 : 0;
    FIRINTR = (intrEn) ? 1 : 0;         //Enable interrupt?
    FIREN = 1;                          //Start FIR accelerator
    //initialize FIR filter with data
    for(i=0; i<=order; i++) {
        FIRWRDATA = data[i];
    }
    //now initialize with coefficients
    for(i=0; i<=order; i++) {
        FIRWRCOEFF = coeff[i];
    }
    return;
}
```

6.2.2 Handling Floating Point Computation

Although the accelerators compute in fixed point arithmetic, it is *not* necessary to convert an entire application from floating point to fixed point in order to make use of the accelerators. Instead, an application can declare variables as floating point, perform floating point operations on them as needed, and convert them into fixed point temporarily for processing by the accelerators.

When converting a floating point variable in C into fixed point, we essentially need to extract its mantissa, while the exponent indicates the scaling factor in the conversion. Fortunately, we can utilize knowledge of the IEEE floating point standard [116] to perform the conversion efficiently. In the following we assume that the C compiler in question represents a "float" data type by the binary32 format in the IEEE standard, but a similar argument applies to different formats as well. The binary32 format consists of a sign bit (s), an 8-bit exponent (x) followed by a 23-bit fraction (m), as shown in Figure 6-5. A number in this representation is equivalent to $(-1)^s \times (1 + m2^{-23}) \times 2^{(x-127)}$. Accordingly, we can construct a 16-bit fixed point representation of a floating point variable by treating the first 16 bits of $(-1)^s \times (1 + m2^{-23})$ as an integer and $2^{-(x-127)}$ as the scaling factor.

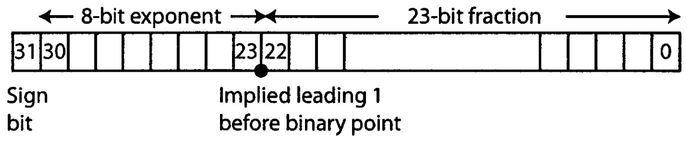


Figure 6-5: IEEE binary32 floating point format.

To make this more clear, we show an example of how to take the square root of a floating point variable using CORDIC.

```
void main() {
    float a, b, c, d;
    ...
    c = a/b;              //c is the result of a floating pt. division
    d = sqrt_cordic(c);
    ...
}

float sqrt_cordic(float x) {
    //Pointers to the ms and ls 16 bits of the float variable x
    unsigned int *float_msb, *float_lsb;
    unsigned int x_fxPtScale, x_fxPt;
    float_lsb = (int*) &x;                      //Set pointers to address of x
    float_msb = (int*) &x;
    float_msb++;

    //Get exponent field and subtract (bias-1)
    x_fxPtScale = ((*float_msb) >> 7) & 0xFF;
    x_fxPtScale -= 126;

    //Get the first 14 bits of the fraction field, then add the implied
    //leading 1
    //Note: this assumes that x is positive (for sqrt)
    x_fxPt = (((*float_msb & 0x7F) + 0x80) << 7) + ((*float_lsb) >> 9);

    //Now program the CORDIC accelerator
    cordic_wrap(x_fxPt, x_fxPtScale, SQRT);

    < reverse the above process to convert CORDIC output back to floating point >
    ...
}
```

6.3 Application Demonstration

Previously we have addressed the operation and energy of the accelerators in isolation. In this section, we will demonstrate the accelerators in the context of complete biomedical applications.

6.3.1 Epileptic Seizure Onset Detection from EEG

The first application aims to detect epileptic seizures from EEG signals acquired on the surface of the scalp [19]. Since seizures are very disruptive to the lives of epilepsy patients, the ability to detect their occurrence in real-time, and raise an alarm to a caregiver, would be extremely valuable. Further, performing the processing on a wearable platform would allow the patient to move freely while being monitored. A high-level view of the algorithm is shown in Figure 6-6, where features are first extracted from a subject's EEG and then analyzed by the classifier to determine whether they resemble features of a normal EEG or those of a seizure event. If a seizure is detected, an alert is raised.

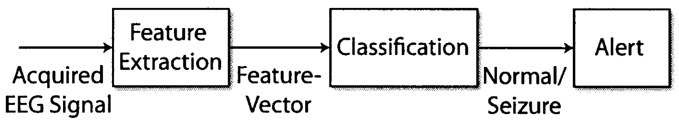


Figure 6-6: High-level overview of the machine learning algorithm for epileptic seizure onset detection [19].

A prototype EEG acquisition system-on-chip targeting this application was demonstrated in [117], and included a custom ASIC with hard-coded filters to implement feature extraction. We also focused on implementing the feature extraction stage of this algorithm, but our platform solution allows the algorithm to be adapted on a patient-specific basis. Additionally, in future work it is also possible to implement the classification stage by leveraging the CORDIC accelerator. Feature extraction involves computing the energy in seven frequency bands of each EEG channel over 2-second windows. As illustrated in Figure 6-7, a channel is first low pass filtered and downsampled by 4. The energy in different bands is estimated with a filter bank followed by magnitude-summation. This is performed for all EEG channels, which number 18 in a common EEG montage, but can be reduced to five by intelligently selecting a subset that contains the key information for detection [118]. After every two seconds, the computed energies of all channels are concatenated into a feature-vector and sent to the classifier.

In an EEG acquisition system (e.g. [23]), sensor nodes for acquiring different EEG channels are distributed around the scalp, but information from all these channels must be aggregated at one location for classification. To eliminate wires between sensor nodes for acquisition and classification, which pose a strangulation hazard, the data is transmitted wirelessly. The amount of data transmitted is greatly reduced by performing local processing of each EEG channel, and only sending the feature-vectors instead of sending the entire captured EEG channel. The amount of data transmitted with and without local processing is given as:

- Local processing (send only feature-vectors): 18 channels * 7 words/2seconds * 16 bits/word = 1008 bits/sec
- No local processing (send entire waveforms): 18 channels * 256 words/second * 16 bits/word = 73728 bits/sec

• Data compression with local processing: 73.1×

Example inputs and outputs computed by the test-chip are shown in Figure 6-8. The top panel plots one pre-recorded EEG channel that is fed into the test-chip. The prerecorded EEG was generously provided by A. Shoeb and was obtained as part of the research in [19]. In the lower panels, each column of 7 points represents a feature-vector computed by the chip, which is then sent to the classifier.

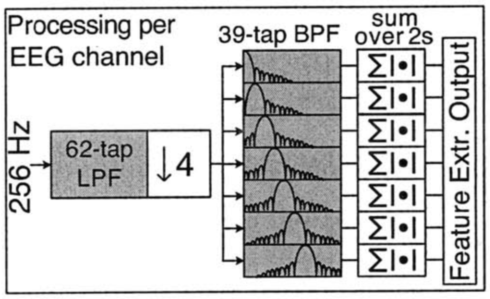


Figure 6-7: Details of the feature extraction portion of the seizure detection algorithm.

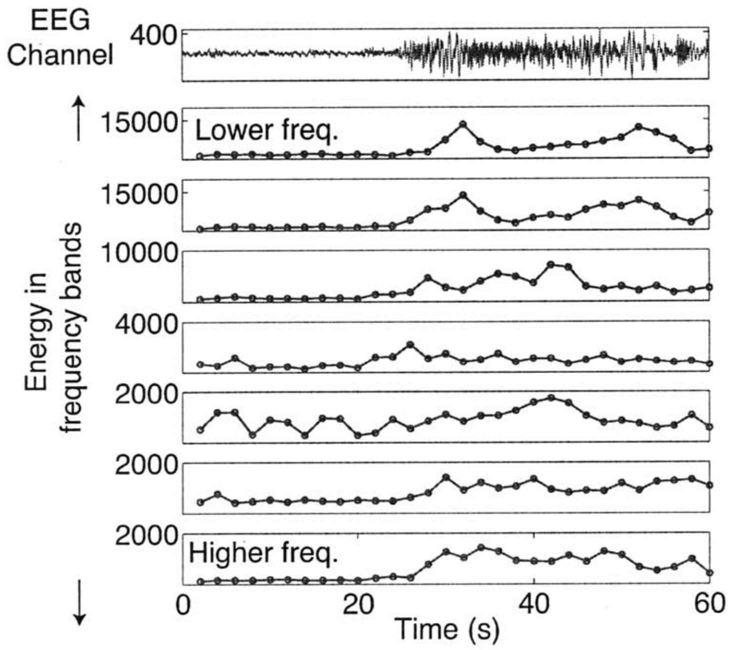


Figure 6-8: One EEG channel and the computed energy in 7 frequency bands.

The shaded blocks in Figure 6-7 indicate portions of the feature extraction, namely the decimation filter and the modulated filter bank, that can take advantage of the FIR accelerator. Since the test-chip only contains one FIR module, the various filters must be handled sequentially. In this case it is advantageous to filter one block of samples at a time to reduce the cost of initializing the FIR module with filter coefficients. Further, note that the filters consist of 62 and 39 taps respectively, larger than 32-word local FIR memory. Consequently, support for this application would not be possible if not for the extended mode feature of the FIR accelerator (Section 2.5.3), which allows 32 taps to be stored in local memory and the remainder to be fetched from main memory. To summarize, Figure 6-9 shows how this algorithm can be mapped onto the biomedical platform. The CPU and DMA control the sequence of events and manage the flow of data. The key processing occurs in the FIR filter, and when it completes one output sample, it raises an interrupt to wake the CPU, which stores the decimated signal in memory or performs magnitude-summation on the band pass filter results.

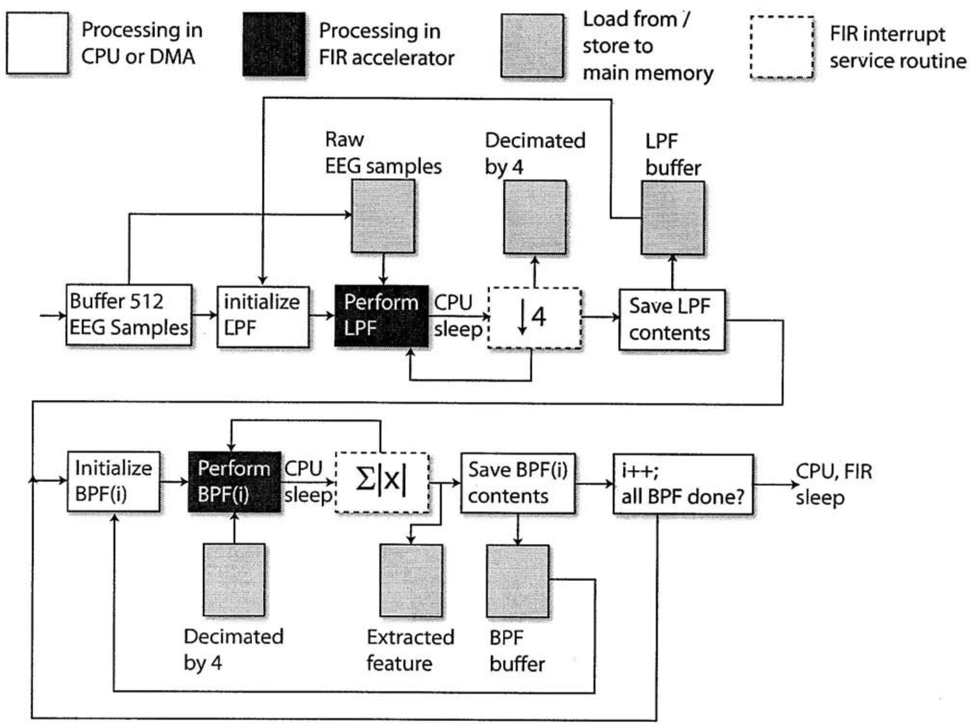


Figure 6-9: Sequence of processing in the feature extraction application.

We now quantify the impact of accelerators in this context. As in Section 6.1.1, we compare the energy of executing the complete application 1) solely on the CPU and multiplier versus 2) using the FIR module for filtering and the CPU for the remaining tasks. Since the accelerated version finishes computation in fewer clock cycles, the platform can operate at a lower V_{DD} while achieving the same latency as the CPU-based version. The measured results are summarized in Table 6.3, showing that the combination of lower cycle count and energy per cycle in the accelerated version contribute to 10.2× savings overall. The energy breakdown between portions of the accelerated version is illustrated in Figure 6-10. The *FIR filtering* category includes the energy consumed by the FIR accelerator and by the transfer of data/coefficients to the accelerator. The *Other* category includes the movement of data needed to time-share one FIR hardware block between different filters in the algorithm. This corresponds to the *initialize* and *save contents* steps in Figure 6-9.

Table 6.3: Measurements of two implementations of the EEG feature extraction algorithm, one utilizing only the CPU and multiplier (CPU-based), and the other with CPU and accelerators (accelerated).

Implementation	V_{DD} (V)	System energy per clock cycle (pJ)	Cycle count per 2s window	Total energy per 2s window (μJ)
CPU-based	1.0	94.2	2099720	198
Accelerated	0.7	49.7	388824	19.3

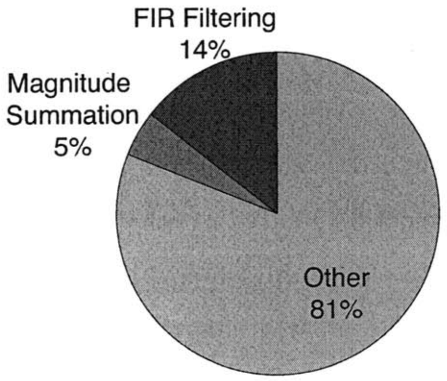


Figure 6-10: Energy breakdown between major portions of the EEG feature extraction algorithm (version employing FIR accelerator).

6.3.2 Detecting the QRS Onset and Duration in an EKG Signal

The second application analyzes an EKG signal to find the onset and duration of the QRS complex, which corresponds to the depolarization of the ventricles of the heart. The amplitude and polarity of the QRS complex depend on the placement of the EKG electrodes, but in many cases it is the most visually obvious part of the signal. The algorithm of interest was proposed in [32] and available as open source software on Physionet [119]. While many

QRS detectors in the literature find the peak of the R wave, the work in [32] locates the beginning of the QRS complex, which can improve the accuracy of heart rate variability analysis. In addition, it finds the duration of the QRS complex, a helpful feature for beat classification. In the context of ambulatory monitoring, these two features can be analyzed in real-time by the biomedical platform to determine if a subject's heart beat is normal or warrants further attention.

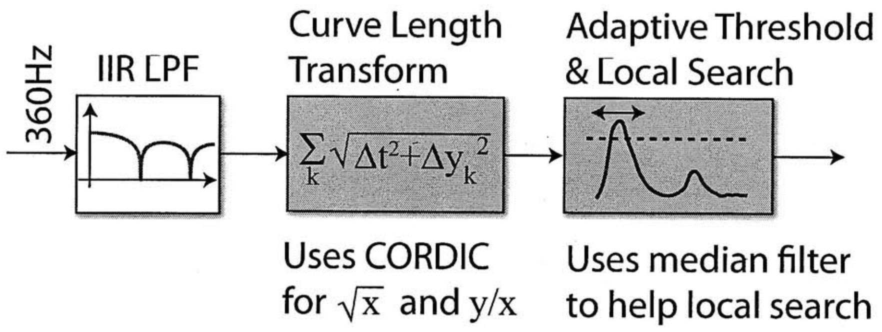

360Hz
IIR LPF
Curve Length Transform
$\sum_k \sqrt{\Delta t^2 + \Delta y_k^2}$
Adaptive Threshold & Local Search
Uses CORDIC for $\sqrt{x}$ and y/x
Uses median filter to help local search

Figure 6-11: Outline of the application for detecting QRS onset and duration in an EKG.

The main components of the algorithm are illustrated in Figure 6-11. The EKG signal is first low pass filtered with a second order IIR low pass filter. Then the curve length transform of the signal is found by computing the arc length of the signal over a sliding window ($\sum_{k=i-w}^{i} \sqrt{\Delta t^2 + \Delta y_k^2}$). The curve length transform accentuates long excursions that are typical of a QRS complex if the window length is similar to the QRS duration. Next, the transformed signal is compared against an adaptive threshold. At the crossing point, the algorithm searches forwards and backwards to locate the start and end of the QRS complex.

The shaded blocks in Figure 6-11 indicate portions of the algorithm that can leverage the accelerators. The curve length transform requires integer division (for scaling) and the square root, which can both be performed with CORDIC. Finally, the last stage of the algorithm requires the minimum and maximum value of the transformed signal near where it crosses the adaptive threshold. This can be aided by the median filter accelerator.

This application brought out several important observations about the CORDIC accelerator design. First, it would be impossible to compute the curve length transform with a conventional CORDIC design due to its limited input range. Only by improving the input range, as discussed in Section 2.3.3, were we able to utilize our CORDIC engine in this computation. Using the modified CORDIC engine, we can compute the square root with 708 cycles (including floating point number conversion described in Section 6.2.2), versus 7440 cycles through software emulation on the CPU, a 10.5× decrease. Second, although the CORDIC engine computes in fixed point and introduces quantization errors as discussed in Section 2.3.3, its accuracy is sufficient for this application because our implementation utilizing CORDIC for both $\sqrt{x}$ and y/x gave final results (the QRS onset and duration) identical to a floating point C implementation.

Shown in Figure 6-12 are segments from two EKG records from the MIT/BIH Arrhythmia Database [119] and the QRS start and end points as computed by the test-chip. We again compare the energy of implementations with and without accelerators. In the following, we constrain the latency for processing one heart beat to 300ms, and adjust the system voltage and frequency accordingly to meet this requirement. Table 6.4 summarizes the measurement results. The use of accelerators provides 11.5× energy savings in this application.

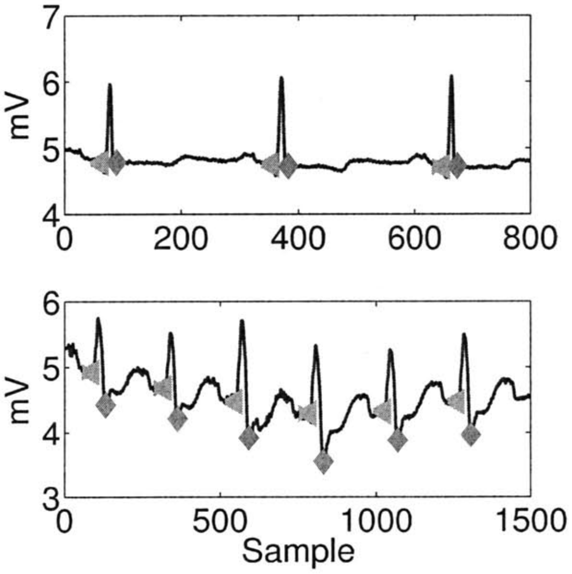


Figure 6-12: Two EKG records annotated with the QRS start and end points as computed by the test-chip.

Figure 6-13 plots the measured energy breakdown as the QRS detection algorithm processes one heart beat. Data for the accelerated version is shown. In this algorithm, the length transform is updated for every input sample, implying the square root is computed for each input as well. Consequently, the square root function is an important portion

Table 6.4: Measurements of two implementations of the QRS detection algorithm, one utilizing only the CPU and multiplier (CPU-based), and the other with CPU and accelerators (accelerated).

Implementation	V_{DD} (V)	System energy per clock cycle (pJ)	Cycle count per beat	Total energy per beat (μJ)
CPU-based	1.0	85.3	2210000	188
Accelerated	0.7	47.3	346000	16.4

contributing 24% of the total energy, even when it is computed efficiently by the CORDIC engine. In the local search phase, the median filter requires 3.8× fewer cycles to find the minimum of a signal compared to searching in software using a loop. However, the local search phase occurs once per heart beat, and thus the overall energy impact of the median filter is small in this particular appplication.

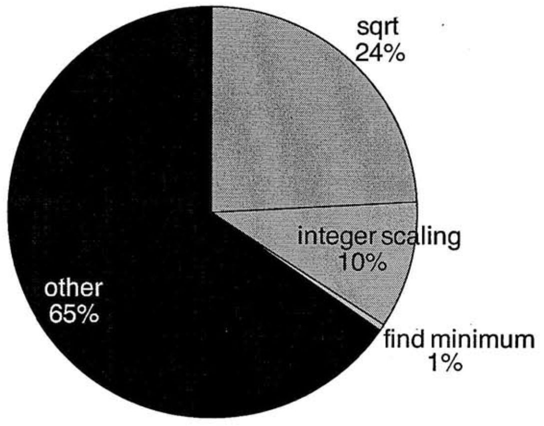


Figure 6-13: Measured energy breakdown between main portions of the QRS detection algorithm (accelerated version).

Chapter 7

Conclusions

This thesis demonstrated an energy-efficient processor for biomedical sensor nodes that can be applied across multiple application domains. This is achieved by desiging for flexibility and low power at different levels of abstraction. At the circuit level, lowering the power supply voltage allows a circuit to operate at its minimum energy point. At low voltages, circuits are much more sensitive to process variation, which becomes increasingly important with process technology scaling. In digital design, the functionality of individual logic gates as well as circuit delays are dramatically affected. However, it was shown in this thesis that both effects can be managed through a careful analysis of failure modes at low voltages and variation-aware design methodologies to address these mechanisms. The resulting ability to operate circuits down to low voltages provides the flexibility of lowering the power supply when a given application does not require high performance.

On the architecture level, power reduction can be achieved without throughput penalty by decreasing unnecessary switching activity. This thesis demonstrated two examples of this concept: reordering computations in an FFT to lower switching activity in the datapath, and using a low-leakage self-timed approach to remove glitches on the SRAM data bus. To improve flexibility, we design the accelerator architectures such that the basic datapaths can be leveraged for various special cases, and this proved useful when we later mapped biomedical applications onto the processor. These special cases are supported primarily by adding appropriate control logic. The resulting accelerators show that this can be achieved with low area cost.

On the processor level, accelerators provide substantial savings in the cycle count and energy of signal processing tasks. The leakage overhead of these blocks is effectively mitigated by module-level power gating. Chip measurements show that accelerators reduce the energy to perform signal processing by two orders of magnitude. Consequently, the energy consumed by the rest of the system in transferring data to and from the accelerators now becomes significant. Accordingly, alternate bus structures or low-swing signaling would be worthwhile to pursue in future work. Nevertheless, by performing key signal processing very efficiently, the biomedical processor in this thesis achieved more than 10× energy savings over two complete biomedical applications compared to a conventional low power processor.

7.1 Summary of Contributions

This thesis focuses on architectures and circuits for energy-efficient digital signal processing, targeting ambulatory medical monitoring as the end application. Details of the contributions are summarized below.

Accelerator architectures

- Identification of signal processing functions that are common in ambulatory medical monitoring applications, and for which hardware acceleration can provide significant energy reduction.

- Analysis of the limitations of the classic CORDIC architecture and proposal of a modified architecture to extend the valid input range and reduce quantization error. The input range was extended by a factor of approximately 2^{14}, and the RMS quantization error decreased by 2-400× depending on the mode of operation.

- Proposal of a control scheme to reduce datapath power in a serial radix-2 FFT architecture, common in low power processors. With this technique, the datapath power is reduced by 50% and the overall FFT power by up to 29%.

- Design of a flexible FIR filter which supports special cases such as high-order filters and polyphase implementations. Filter demonstrates a latch-based register file which reduces the overall FIR power by 31%.

- Design and integration of the above accelerators in a prototype biomedical signal processing platform in $0.13\mu m$ CMOS.

- Demonstration of ambulatory monitoring applications on the processing platform, and most importantly, mapping key signal processing functions to the accelerators. Using the accelerators to aid the CPU enables greater than 10× energy reduction in complete EEG and EKG applications compared to implementations on a conventional processor.

Low-voltage logic and SRAM design

- Proposal of design flow and timing verification methodology for two classes of low-voltage logic: 1) targeting deep sub-threshold operation at the minimum energy point, and 2) operating from near transistor threshold to nominal V_{DD}, to support commensurate performance demands.

- Demonstration of a 65nm microcontroller which operates down to 0.3V. This represents the first processor to achieve deep sub-threshold operation at the 65nm node, and the first to incorporate standard cell design and timing methodology explicitly considering local variation.

- Identification of the energy to drive the SRAM internal data bus as a large contributor to total energy in low-voltage SRAMs. Accordingly, a self-timed scheme to reduce glitches on the data bus is employed. In contrast to the usual practice of designing for the worst case, the proposed scheme removes most glitches in the average case at a small fraction of the leakage cost.

- Design and integration of a 0.5V-1V SRAM as the main memory and FFT memory of the biomedical processor.

System architecture

- Analysis of the energy of interfacing functional modules to a typical low power processor architecture. It is shown that optimizing data transfers between modules will have a large impact on the overall system energy.

- Analysis of a dual-bus structure as an alternative to the common single-bus architecture. Proposal of a framework to assign modules to the two buses in order to minimize bus energy while accounting for the bus transaction characteristics of the end applications.

7.2 Future Work

This thesis investigated several aspects of low power processor design for biomedical applications, but clearly there are many other interesting avenues for further exploration. We list some directions for future work below.

Application-Specific Instruction-Set Processors

This thesis focused on a top-level architecture for the processing platform consisting of a general-purpose RISC CPU augmented by specialized functional modules. An alternate architecture is that of application-specific instruction-set processors (ASIPs). In ASIPs, a general-purpose instruction-set is extended by specialized instructions targeted towards a specific application [120]. These specialized instructions are supported by functional units – in the example given in [120], an ASIP for Reed-Solomon encoding is extended by a module for computing Galois-field multiplications. This is akin to the addition of accelerators explored in this thesis, but an ASIP provides custom instructions for their usage as well as compiler support for these instructions. On the other hand, extending the instruction set would likely increase the size and complexity of the processor. It would be worthwhile to evaluate an ASIP against a traditional CPU/peripheral architecture in the context of biomedical applications in order to understand the performance and energy implications.

Compiler Support for Accelerators

Accelerators are only useful to the extent that they can be leveraged by the software programmer. In the accelerator design, we have made an effort to simplify their usage and support a variety of special cases. Nevertheless, it would greatly aid the programmer's task if a compiler can map a high-level description of an algorithm onto both the CPU and accelerators. For a module like CORDIC which takes an input argument and provides an output, this may be relatively simple. It is much more challenging to automatically generate code that allows one hardware module to be time-shared across different portions of the software (e.g., allowing one FIR module to support several filters).

Multi-Function Accelerator

The accelerators in this work are each targeted towards a specific function. However, the FFT and FIR filter, and likely other potential accelerators, rely on a common set of arithmetic components. Instead of implementing separate modules, one can consider designing a multi-function accelerator that can support both FIR filtering and FFT, consisting of adders, multipliers, a local memory, and control logic managing the flow of data. The control aspect of such a multi-function accelerator brings forth interesting design problems and trade-offs.

Energy of FFT Architectures

Due to the importance of the FFT in signal processing, numerous FFT architectures have been proposed in previous work. Different designs (e.g. radix-2, radix-4, radix-2^3) are often evaluated based on their peak throughput or the utilization rate of the datapath components. To our knowledge, no prior work has compared architectures based on energy. Differences between FFT architectures are somewhat more complex than a simple replication of components. For example, a radix-2 butterfly contains one complex multiplier while a radix-4 butterfly contains three complex multipliers. However, the number of butterflies in radix-2 and radix-4 are $N/2 \log_2 N$ and $N/4 \log_4 N$ respectively. A study of the trade-offs between a larger datapath and fewer cycles to compute an FFT would be informative for future sensor processor design.

Process Technology Selection

High performance, high volume products such as microprocessors and mobile chipsets are implemented on the most advanced process technologies, mainly to take advantage of the lower area and cost per chip enabled by process scaling. For other applications that do not require high performance nor achieve high volumes, such as biomedical sensor nodes, we may instead select a technology to minimize energy given application constraints.

In Section 4.1.2 we examined how to select a process technology to minimize energy considering the frequency constraints and duty cycle of an application. This was accomplished by simulating a circuit across different supply voltages at each technology node of interest. While simulating the exact circuit gives the most accurate results, an approximate approach that reduces the simulation time would be highly desirable, particularly for large circuits. At the end of Section 4.1.2 we outlined one such approach that involves abstracting a circuit into parameters such as the average switched capacitance and the leakage current, then scaling these parameters across process technologies. Our preliminary simulations indicate that it is challenging to accurately model how the leakage of a circuit scales with supply voltage across different technology nodes; models using simulations of an inverter do not suffice. Refining the approach into a reasonably accurate model would be helpful to circuit and system designers.

Appendix A

Cycle Count Optimizations in the FIR Filter

A.1 Symmetric/Anti-Symmetric Filters

Symmetric and anti-symmetric filters are a popular class of filters due to their linear phase characteristics. The impulse response of a symmetric FIR filter satisfies the relation $h[N-n] = h[n]$, where N is an integer. Because the filter coefficients are symmetric about $N/2$, it is well-known that the convolution of Equation 2.17 can be rewritten as:

$$y[n] = \sum_{k=0}^{(N-1)/2} h[k](x[n-k] + x[n-N+k]), \text{N odd} \tag{A.1}$$

$$y[n] = \sum_{k=0}^{N/2-1} h[k](x[n-k] + x[n-N+k]) + h[N/2]x[n-N/2], \text{N even} \tag{A.2}$$

This type of filter can be implemented with half the number of multiplications compared to a non-symmetric filter, which in our serial architecture translates into half the number of clock cycles. To compute the above convolution in the FIR accelerator, we fetch two words from the local data memory per clock cycle and add or subtract them before multiplying with a coefficient.

A.2 High-Order Filters

Due to area constraints, the local data and coefficient memory of the FIR filter is restricted to 32 words long. Typically this would imply that only filters of order ≤ 31 can take advantage of the accelerator. However, to introduce additional flexibility, we design the control logic to accommodate filters of order up to 127. When the filter order is more than 31, 32 pairs of data and coefficients are stored in the local memory and the rest is kept in main memory. The filter first performs multiply-accumulate on the pairs stored locally, then fetch the remaining pairs from main memory in order to compute one output sample.

Although we incur more clock cycle and active energy with this approach than if all data/coefficient pairs were stored locally, we still achieve savings during part of the computations.

A.3 Context Switching

Many applications require more than one FIR filter to appropriately process an incoming signal. Accordingly, we designed an operating mode in which the data and coefficients of two filters are stored in the local memory, and either one can be selected as the active filter at any time. This provides savings in energy and clock cycles that would otherwise be needed to initialize the local memory with another filter. Due to the size of the local memory, this mode supports filter orders of ≤ 15; however, the two filters can be of different orders.

A.4 Polyphase Implementation

We further extend the context switching mode to perform a polyphase implementation of a decimation filter. Although the accelerator design supports decimation-by-2, the concept can be easily extended to decimation by higher factors. Figure A-1 shows the straightforward approach to form a decimation-by-2 filter. However, this approach is wasteful since the filter runs at the higher input sample rate, but every other output sample is discarded. In this approach, each output sample requires $2(N+1)$ multiplications.

Instead, it is more efficient to implement the filter in its polyphase decomposition as shown in Figure A-2, where $H_{even}(z)$ and $H_{odd}(z)$ denote two filters of order $(N+1)/2$,

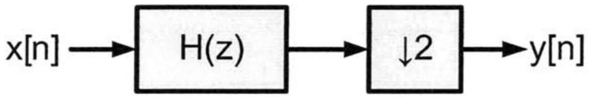


Figure A-1: Conceptual diagram of a straightforward decimation-by-2 filter. Half of the computations are wasted since every second output sample is discarded.

consisting of the even and odd coefficients of $H(z)$ respectively. The top branch filters the even samples of $x[n]$ with $H_{even}(z)$ and the bottom branch filters the odd samples; after both branches have processed one input, the results are added to form one output sample. It can be shown that Figure A-1 and Figure A-2 are mathematically equivalent [59]. Moreover, since $H_{even}(z)$ and $H_{odd}(z)$ each perform $(N + 1)/2$ multiplications at a rate of $f_{in}/2 = f_{out}$, the total number of multiplications per output is only $(N + 1)$. This represents half as many multiplications as in the straightforward approach, which again translates into a $2\times$ reduction in the cycle count in our serial architecture.

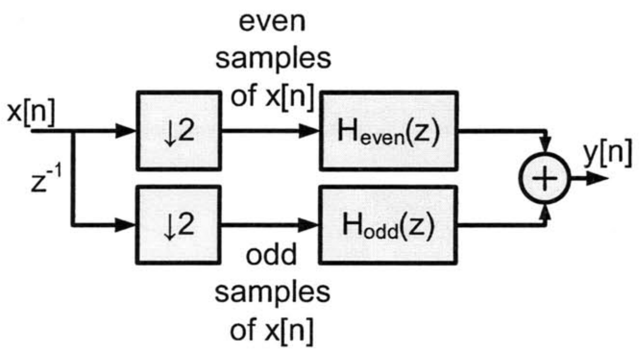


Figure A-2: Conceptual diagram of polyphase implementation of a decimation-by-2 filter.

In the FIR accelerator, we leverage the context switching mode to realize Figure A-2. The accelerator stores $H_{even}(z)$ and $H_{odd}(z)$ as two different filters in local memory, automatically switches between them after every input sample, and adds the results appropriately.

A.5 Multiplication by Window

In addition to performing convolution, we can leverage the multiplier and local memory to multiply a block of data with a window function. This type of operation arises often in spectral analysis where a block of time-domain samples are multiplied by a window function before undergoing the DFT. Different types of window functions (for example Hamming windows or Kaiser windows) allow trading off frequency resolution and spectral leakage in spectral analysis. We support this efficiently by storing the window within local memory and multiplying input samples with successive values in the window, thus saving the energy of repeatedly accessing the window from main memory.

Bibliography

[1] R. Yazicioglu, T. Torfs, J. Penders, I. Romero, H. Kim, P. Merken, B. Gyselinckx, H. Yoo, and C. Van Hoof, "Ultra-low-power wearable biopotential sensor nodes," in *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, Sept. 2009, pp. 3205 –3208.

[2] A.-W. Wong, D. McDonagh, G. Kathiresan, O. Omeni, O. El-Jamaly, T.-K. Chan, P. Paddan, and A. Burdett, "A 1V, micropower system-on-chip for vital-sign monitoring in wireless body sensor networks," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, 3-7 2008, pp. 138 –602.

[3] Mindray NetGuard. [Online]. Available: http://www.na.mindray.com/products/netguard.html

[4] Corventis PiiX^{TM}. [Online]. Available: http://corventis.com/US/medprof.asp

[5] N. Verma, A. Shoeb, J. V. Guttag, and A. P. Chandrakasan, "A micro-power EEG acquisition SoC with integrated seizure detection processor for continuous patient monitoring," in *Symposium on VLSI Circuits (VLSI) Digest of Technical Papers*, June 2009, pp. 62–63.

[6] T. Denison, W. Santa, R. Jensen, D. Carlson, G. Molnar, and A.-T. Avestruz, "An $8\mu W$ heterodyning chopper amplifier for direct extraction of 2μVrms neuronal biomarkers," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2008, pp. 162–603.

[7] N. Verma and A. Chandrakasan, "A 25μW 100kS/s 12b ADC for wireless micro-sensor applications," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2006, pp. 822–831.

[8] A. Agnes, E. Bonizzoni, P. Malcovati, and F. Maloberti, "A 9.4-ENOB 1V 3.8μW 100kS/s SAR ADC with time-domain comparator," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2008, pp. 246–610.

[9] ChipCon 2550 Datasheet. [Online]. Available: http://focus.ti.com/lit/ds/symlink/cc2550.pdf

[10] R. F. Yazicioglu, P. Merken, R. Puers, and C. Van Hoof, "A 200 μW eight-channel EEG acquisition ASIC for ambulatory EEG systems," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 12, pp. 3025–3038, Dec. 2008.

[11] J. L. Bohorquez, M. Yip, A. P. Chandrakasan, and J. L. Dawson, "A digitally-assisted sensor interface for biomedical applications," in *Symposium on VLSI Circuits (VLSI) Digest of Technical Papers*, 2010, to be published.

[12] INA333 Datasheet. [Online]. Available: http://focus.ti.com/lit/ds/symlink/ina333.pdf

[13] ADS7866 Datasheet. [Online]. Available: http://focus.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=ads7866&fileType=pdf

[14] S. C. Jocke, J. F. Bolus, S. N. Wooters, A. D. Jurik, A. C. Weaver, T. N. Blalock, and B. H. Calhoun, "A 2.6-μW sub-threshold mixed-signal ECG SoC," in *Symposium on VLSI Circuits (VLSI) Digest of Technical Papers*, June 2009, pp. 60–61.

[15] M. W. Phyu, Y. Zheng, B. Zhao, L. Xin, and Y. S. Wang, "A real-time ECG QRS detection ASIC based on wavelet multiscale analysis," in *IEEE Asian Solid-State Circuits Conference*, Nov. 2009, pp. 293–296.

[16] MSP430F5438 Datasheet. [Online]. Available: http://focus.ti.com/lit/ds/symlink/msp430f5438.pdf

[17] P. Mercier, D. Daly, and A. Chandrakasan, "An energy-efficient all-digital UWB transmitter employing dual capacitively-coupled pulse-shaping drivers," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 6, pp. 1679–1688, June 2009.

[18] Metronic Reveal. [Online]. Available: http://www.medtronic.com/physician/reveal/index.html#

[19] A. Shoeb, H. Edwards, J. Connolly, B. Bourgeois, S. T. Treves, and J. Guttag, "Patient-specific seizure onset detection," *Epilepsy & Behavior*, vol. 5, no. 4, pp. 483–498, 2004.

[20] G. B. Moody and R. G. Mark, "A database to support development and evaluation of intelligent intensive care monitoring," in *Computers in Cardiology*, Sept. 1996, pp. 657–660.

[21] A. Salarian, H. Russmann, C. Wider, P. Burkhard, F. Vingerhoets, and K. Aminian, "Quantification of tremor and bradykinesia in Parkinson's Disease using a novel ambulatory monitoring system," *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 2, pp. 313–322, Feb. 2007.

[22] A. Buizza, G. Coldani, G. Danese, R. Gandolfi, P. Ghidetti, and R. Lombardi, "An instrument with a DSP for monitoring human biological parameters," in *IEEE International Conference on Electronics, Circuits and Systems*, vol. 2, 2001, pp. 757–760.

[23] A. Shoeb, J. Guttag, S. Schachter, D. Schomer, B. Bourgeois, and S. Treves, "Detecting seizure onset in the ambulatory setting: Demonstrating feasibility," in *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, Jan. 2005, pp. 3546–3550.

[24] S. Hanson, B. Zhai, M. Seok, B. Cline, K. Zhou, M. Singhal, M. Minuth, J. Olson, L. Nazhandali, T. Austin, D. Sylvester, and D. S. Blaauw, "Performance and

variability optimization strategies in a sub-200mV, 3.5pJ/inst, 11nW subthreshold processor," in *Symposium on VLSI Circuits (VLSI) Digest of Technical Papers*, June 2007, pp. 152–153.

[25] B. Zhai, L. Nazhandali, J. Olson, A. Reeves, M. Minuth, R. Helfand, S. Pant, D. Blaauw, and T. Austin, "A 2.60pJ/Inst subthreshold sensor processor for optimal energy efficiency," in *Symposium on VLSI Circuits (VLSI) Digest of Technical Papers*, June 2006, pp. 154–155.

[26] J. Kwong, Y. Ramadass, N. Verma, M. Koesler, K. Huber, H. Moormann, and A. Chandrakasan, "A 65nm Sub-V_t microcontroller with integrated SRAM and switched-capacitor DC-DC converter," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2008, pp. 318–319.

[27] M. Seok, S. Hanson, Y.-S. Lin, Z. Foo, D. Kim, Y. Lee, N. Liu, D. Sylvester, and D. Blaauw, "The Phoenix Processor: A 30pW platform for sensor applications," *IEEE Symposium on VLSI Circuits*, pp. 188–189, June 2008.

[28] N. Ickes, D. Finchelstein, and A. Chandrakasan, "A 10-pJ/instruction, 4-MIPS micropower DSP for sensor applications," in *IEEE Asian Solid-State Circuits Conference (A-SSCC) Digest of Technical Papers*, Nov. 2008, pp. 289–292.

[29] Z. Nie, L. Wang, W. Chen, T. Zhang, and Y. Zhang, "A low power biomedical signal processor ASIC based on hardware software codesign," in *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, Sept. 2009, pp. 2559–2562.

[30] H. Kim, R. Yazicioglu, T. Torfs, P. Merken, C. Van Hoof, and H.-J. Yoo, "An integrated circuit for wireless ambulatory arrhythmia monitoring systems," in *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, Sept. 2009, pp. 5409–5412.

[31] A. Ruha, S. Sallinen, and S. Nissila, "A real-time microprocessor QRS detector system with a 1-ms timing accuracy for the measurement of ambulatory HRV," *IEEE Transactions on Biomedical Engineering*, vol. 44, no. 3, pp. 159 –167, Mar. 1997.

[32] W. Zong, G. Moody, and D. Jiang, "A robust open-source algorithm to detect onset and duration of QRS complexes," in *Computers in Cardiology (CinC) Digest of Technical Papers*, Sept. 2003, pp. 737–740.

[33] P. S. Hamilton and W. J. Tompkins, "Quantitative investigation of QRS detection rules using the MIT/BIH Arrhythmia Database," *IEEE Transactions on Biomedical Engineering*, vol. BME-33, no. 12, pp. 1157–1165, Dec. 1986.

[34] M. Tsuruoka, Y. Tsuruoka, R. Shibasaki, and Y. Yasuoka, "Analysis of 1/f fluctuations of heart rate response while walking or listening to sounds," in *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, Aug. 2007, pp. 5915 –5918.

[35] Open source arrhythmia detection software. [Online]. Available: http://www.eplimited.com/confirmation.htm

[36] G. Ravindran and O. Ashwin Chandar, "Ambulatory electroencephalographic monitoring system for the analysis of brain waves," in *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, vol. 1, Sept. 2004, pp. 345–348.

[37] C. Papadelis, C. Kourtidou-Papadeli, P. Bamidis, I. Chouvarda, D. Koufogiannis, E. Bekiaris, and N. Maglaveras, "Indicators of sleepiness in an ambulatory EEG study of night driving," in *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) Digest of Technical Papers*, Sept. 2006, pp. 6201–6204.

[38] H.-J. Park, D.-U. Jeong, and K.-S. Park, "Automated detection and elimination of periodic ECG artifacts in EEG using the energy interval histogram method," *IEEE Transactions on Biomedical Engineering*, vol. 49, no. 12, pp. 1526–1533, Dec. 2002.

[39] A Single-Chip Pulsoximeter Design Using the MSP430. [Online]. Available: http://www.ti.com/litv/pdf/slaa274

[40] Z. H. Syed, "MIT Automated Auscultation System," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[41] D. P. W. Ellis. (2002) Phase vocoder in MATLAB. [Online]. Available: http://labrosa.ee.columbia.edu/matlab/pvoc/

[42] N. Ickes, "A Micropower DSP for Sensor Applications," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2008.

[43] D. S. Richards, "VLSI median filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 38, no. 1, pp. 145–153, Jan. 1990.

[44] C. J. Tsai, E. H. Lu, C. H. Chen, J. Y. Lee, and I. C. Jou, "A new architecture of median filters with linear hardware complexity," in *IEEE International Symposium on Circuits and Systems (ISCAS) Digest of Technical Papers*, vol. 1, June 1991, pp. 101–103.

[45] S. B. Leeb, A. Ortiz, R. F. Lepard, S. R. Shaw, and J. L. Kirtley, "Applications of real-time median filtering with fast digital and analog sorters," *IEEE/ASME Transactions on Mechatronics*, vol. 2, no. 2, pp. 136–143, June 1997.

[46] K. Oflazer, "Design and implementation of a single-chip 1-D median filter," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 31, no. 5, pp. 1164–1168, Oct. 1983.

[47] Z. Vasicek and L. Sekanina, "Novel hardware implementation of adaptive median filters," *11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pp. 1–6, Apr. 2008.

[48] A. Hiasat, M. Al-Ibrahim, and K. Gharaibeh, "Design and implementation of a new efficient median filtering algorithm," *IEE Proceedings - Vision, Image and Signal Processing*, vol. 146, no. 5, pp. 273–278, Oct. 1999.

[49] K. Benkrid, D. Crookes, and A. Benkrid, "Design and implementation of a novel algorithm for general purpose median filtering on FPGAs," in *IEEE International*

Symposium on Circuits and Systems (ISCAS) Digest of Technical Papers, vol. 4, Aug. 2002.

[50] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Prentice Hall, 2003.

[51] J. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. EC-8, pp. 330–334, Sept. 1959.

[52] J. S. Walther, "A unified algorithm for elementary functions," in *Spring Joint Computer Conference*, 1971, pp. 379–385.

[53] P. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, Sept. 2009.

[54] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *FPGA*, 1998, pp. 191–200.

[55] X. Hu, R. Harber, and S. Bass, "Expanding the range of convergence of the CORDIC algorithm," *IEEE Transactions on Computers*, vol. 40, no. 1, pp. 13–21, Jan. 1991.

[56] R. Harber, J. Li, X. Hu, and S. Bass, "Bit-serial CORDIC circuits for use in a VLSI silicon compiler," in *IEEE International Symposium on Circuits and Systems*, vol. 1, 8-11 1989, pp. 154–157.

[57] H. Hahn, D. Timmermann, B. Hosticka, and B. Rix, "A unified and division-free CORDIC argument reduction method with unlimited convergence domain including inverse hyperbolic functions," *IEEE Transactions on Computers*, vol. 43, no. 11, pp. 1339–1344, Nov. 1994.

[58] Y. H. Hu, "The quantization effects of the CORDIC algorithm," *IEEE Transactions on Signal Processing*, vol. 40, no. 4, pp. 834–844, Apr. 1992.

[59] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed. Prentice-Hall, 1999.

[60] A. Wang and A. P. Chandrakasan, "Energy-aware architectures for a real-valued FFT implementation," in *International Symposium on Low-Power Electronics and Design (ISLPED) Digest of Technical Papers*, Aug. 2003, pp. 360–365.

[61] D. Cohen, "Simplified control of FFT hardware," vol. 24, no. 6, pp. 577–579, Dec 1976.

[62] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE Transactions on Signal Processing*, vol. 48, no. 3, pp. 917–921, Mar. 2000.

[63] M. Hasan and T. Arslan, "Implementation of low-power FFT processor cores using a novel order-based processing scheme," *IEE Proceedings – Circuits, Devices and Systems*, vol. 150, no. 3, pp. 149–154, June 2003.

[64] ——, "A coefficient memory addressing scheme for VLSI implementation of FFT processors," in *IEEE International Symposium on Circuits and Systems*, vol. 4, 2002, pp. 850–853.

[65] M. C. Pease, "Organization of large scale Fourier processors," *Journal of the ACM*, vol. 16, no. 3, pp. 474–482, 1969.

[66] C. Nicol and P. Larsson, "Low power multiplication for FIR filters," in *International Symposium on Low-Power Electronics and Design (ISLPED) Digest of Technical Papers*, Aug. 1997, pp. 76–79.

[67] Atmel 8-bit Microcontroller with 2K Bytes Flash AT89C2051 datasheet. [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/doc0368.pdf

[68] MSP430FG4619 Datasheet. [Online]. Available: http://focus.ti.com/docs/prod/folders/print/msp430fg4619.html

[69] Microchip PIC18F1220/1320 Datasheet. [Online]. Available: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010264#1

[70] Cortex^{TM}-M3 Technical Reference Manual. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0337h/index.html

[71] B. Calhoun, F. Honore, and A. Chandrakasan, "Design methodology for fine-grained leakage control in MTCMOS," in *International Symposium on Low-Power Electronics and Design (ISLPED) Digest of Technical Papers*, Aug. 2003, pp. 104–109.

[72] S. Bhunia, N. Banerjee, Q. Chen, H. Mahmoodi, and K. Roy, "A novel synthesis approach for active leakage power reduction using dynamic supply gating," in *ACM/IEEE Design Automation Conference (DAC) Digest of Technical Papers*, June 2005, pp. 479–484.

[73] E. Vittoz and J. Fellrath, "CMOS analog integrated circuits based on weak-inversion operation," *IEEE Journal of Solid-State Circuits*, vol. SC-12, pp. 224–231, June 1977.

[74] E. A. Vittoz, "The electronic watch and low-power circuits," *Solid-State Circuits Society Newsletter*, vol. 13, no. 3, pp. 7–23, 2008.

[75] A. Wang, A. Chandrakasan, and S. Kosonocky, "Optimal supply and threshold scaling for sub-threshold CMOS circuits," in *IEEE Computer Society Annual Symposium on VLSI*, Apr. 2002, pp. 5–9.

[76] B. Calhoun, A. Wang, and A. Chandrakasan, "Modeling and sizing for minimum energy operation in subthreshold circuits," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 9, pp. 1778–1786, Sept. 2005.

[77] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm early design exploration," *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816–2823, Nov. 2006.

[78] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, 2009.

[79] BSIM4 Manual. [Online]. Available: http://www-device.eecs.berkeley.edu/bsim3/~bsim4.html

[80] B. Zhai, S. Hanson, D. Blaauw, and D. Sylvester, "Analysis and mitigation of variability in subthreshold design," in *International Symposium on Low-Power Electronics and Design (ISLPED) Digest of Technical Papers*, Aug. 2005, pp. 20–25.

[81] A. Wang and A. Chandrakasan, "A 180-mV subthreshold FFT processor using a minimum energy design methodology," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 1, pp. 310–319, Jan. 2005.

[82] B. H. Calhoun, A. Wang, and A. Chandrakasan, "Device Sizing for Minimum Energy Operation in Subthreshold Circuits," in *Custom Integrated Circuits Conference (CICC) Digest of Technical Papers*, Oct. 2004, pp. 95–98.

[83] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers, "Matching properties of MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1439, Oct. 1989.

[84] K. Gettings and D. Boning, "Study of CMOS process variation by multiplexing analog characteristics," *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, no. 4, pp. 513–525, Nov. 2008.

[85] J. Kwong and A. Chandrakasan, "Variation-driven device sizing for minimum energy sub-threshold circuits," in *International Symposium on Low-Power Electronics and Design (ISLPED) Digest of Technical Papers*, Oct. 2006, pp. 8–13.

[86] J. Kwong, "A sub-threshold cell library and methodology," Master's thesis, Massachusetts Institute of Technology, 2006.

[87] *Synopsys® Timing Constraints and Optimization User Guide, Edition D-2010.03.*

[88] N. C. Beaulieu and F. Rajwani, "Highly accurate simple closed-form approximations to lognormal sum distributions and densities," in *IEEE Communications Letters*, vol. 8, no. 12, Dec. 2004, pp. 709–711.

[89] S. Schwartz and Y. Yeh, "On the distribution function and moments of power sums with log-normal components," *Bell Sys. Tech. Journal*, vol. 61, no. 7, pp. 1441–1462, Sept. 1982.

[90] Y. Tsividis, *Operation and Modeling of the MOS Transistor.* Oxford University Press, 1999.

[91] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, "A 320mV 56μW 411GOPS/Watt ultra-low voltage motion estimation accelerator in 65nm CMOS," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2008, pp. 316–317.

[92] B. H. Calhoun and A. P. Chandrakasan, "A 256-kb 65-nm sub-threshold SRAM design for ultra-low-voltage operation," in *IEEE Journal of Solid-State Circuits*, vol. 42, no. 3, Mar. 2007, pp. 680–688.

[93] R. Hegde and N. Shanbhag, "Soft digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 6, pp. 813–823, Dec. 2001.

[94] ——, "A voltage overscaled low-power digital filter IC," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 2, pp. 388–391, Feb. 2004.

[95] B. Shim, S. Sridhara, and N. Shanbhag, "Reliable low-power digital signal processing via reduced precision redundancy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, pp. 497–510, May 2004.

[96] B. Shim and N. Shanbhag, "Energy-efficient soft error-tolerant digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 4, pp. 336–348, 2006.

[97] N. Verma and A. P. Chandrakasan, "A 256-kb 65 nm 8T subthreshold SRAM employing sense-amplifier redundancy," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 141–149, Jan. 2008.

[98] Y. K. Ramadass and A. P. Chandrakasan, "Voltage scalable switched capacitor DC-DC converter for ultra-low-power on-chip applications," in *Power Electronics Specialists Conference*, June 2007, pp. 2353–2359.

[99] N. Verma, "Ultra-Low-Power SRAM Design In High Variability Advanced CMOS," Ph.D. dissertation, Massachusetts Institute of Technology, 2009.

[100] K. Takeda, Y. Hagihara, Y. Aimoto, M. Nomura, Y. Nakazawa, T. Ishii, and H. Kobatake, "A read-static-noise-margin-free SRAM cell for low-Vdd and high-speed applications," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, 2005, pp. 478–611.

[101] B. Calhoun and A. Chandrakasan, "A 256-kbit Sub-threshold SRAM 65nm CMOS," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2006, pp. 628–629.

[102] J. Chen, L. T. Clark, and T.-H. Chen, "An ultra-low-power memory with a subthreshold power supply voltage," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 10, pp. 2344–2353, Oct. 2006, verified (IEEE Proc).

[103] T.-H. Kim, J. Liu, J. Keane, and C. H. Kim, "A high-density subthreshold SRAM with data-independent bitline leakage and virtual ground replica scheme," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2007, pp. 330–331.

[104] B. Zhai, D. Blaauw, D. Sylvester, and S. Hanson, "A sub-200mV 6T SRAM in $0.13\mu m$ CMOS," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2007, pp. 332–333.

[105] N. Verma and A. Chandrakasan, "A 65nm 8T sub-V_t SRAM employing sense-amplifier redundancy," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2007, pp. 328–329.

[106] L. Chang, Y. Nakamura, R. K. Montoye, J. Sawada, A. K. Martin, K. Kinoshita, F. H. Gebara, K. B. Agarwal, D. J. Acharyya, W. Haensch, K. Hosokawa, and D. Jamsek, "A 5.3GHz 8T-SRAM with operation down to 0.41V in 65nm CMOS," in *Symposium on VLSI Circuits (VLSI) Digest of Technical Papers*, 2007, pp. 252–253.

[107] J. Kulkarni, K. Kim, and K. Roy, "A 160 mV robust schmitt trigger based subthreshold SRAM," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 10, pp. 2303 –2313, Oct. 2007.

[108] I. J. Chang, J.-J. Kim, S. Park, and K. Roy, "A 32kb 10T subthreshold SRAM array with bit-interleaving and differential read scheme in 90nm CMOS," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2008, pp. 388–622.

[109] M. Sinangil, N. Verma, and A. Chandrakasan, "A reconfigurable 65nm SRAM achieving voltage scalability from 0.25-1.2V and performance scalability from 20kHz-200MHz," in *IEEE European Solid-State Circuits Conference (ESSCIRC) Digest of Technical Papers*, Sept. 2008.

[110] ——, "A 45nm 0.5V 8T column-interleaved SRAM with on-chip reference selection loop for sense-amplifier," in *IEEE Asian Solid-State Circuits Conference (A-SSCC) Digest of Technical Papers*, Feb. 2009, pp. 225–228.

[111] E. Seevinck, F. List, and J. Lohstroh, "Static noise margin analysis of MOS SRAM cells," *IEEE Journal of Solid-State Circuits*, vol. SC-22, no. 5, pp. 748–754, Oct. 1987.

[112] L. Chang, R. Montoye, Y. Nakamura, K. Batson, R. Eickemeyer, R. Dennard, W. Haensch, and D. Jamsek, "An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 956–963, Apr. 2008.

[113] K. Takeda, H. Ikeda, Y. Hagihara, M. Nomura, and H. Kobatake, "Redefinition of write margin for next-generation SRAM and write-margin monitoring circuit," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, Feb. 2006, pp. 2602–2611.

[114] V. Sze and A. Chandrakasan, "A 0.4-V UWB baseband processor," in *International Symposium on Low-Power Electronics and Design (ISLPED) Digest of Technical Papers*, Aug. 2007, pp. 262–267.

[115] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[116] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–58, Aug. 2008.

[117] N. Verma, A. Shoeb, J. Bohorquez, J. Dawson, J. Guttag, and A. P. Chandrakasan, "A micro-power EEG acquisition SoC with integrated feature extraction processor for a chronic seizure detection system," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 804–816, Apr. 2010.

[118] E. I. Shih, A. H. Shoeb, and J. V. Guttag, "Sensor selection for energy-efficient ambulatory medical monitoring," in *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services.* New York, NY, USA: ACM, 2009, pp. 347–358.

[119] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000 (June 13), circulation Electronic Pages: http://circ.ahajournals.org/cgi/content/full/101/23/e215.

[120] G. Goossens, P. Dytrych, and D. Lanneer, *Low-Power Electronics Design*, C. Piguet, Ed. CRC Press, 2004.