ENERGY-EFFICIENT PROCESSOR SYSTEM DESIGN

by

Thomas David Burd

Memorandum No. UCB/ERL M01/13

7 March 2001

ENERGY-EFFICIENT PROCESSOR SYSTEM DESIGN

by

Thomas David Burd

Memorandum No. UCB/ERL M01/13

7 March 2001

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Energy-Efficient Processor System Design

by

Thomas David Burd

B.S. (University of California, Berkeley) 1992
M.S. (University of California, Berkeley) 1994

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy
in

Engineering - Electrical Engineering
and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Robert W. Brodersen, Chair
Professor Borivoje Nikolic
Professor Paul K. Wright

Spring 2001

The dissertation of Thomas David Burd is approved:

Chair	Date
	Date
	Date

University of California, Berkeley

Spring 2001

Energy Efficient Processor System Design

by

Thomas David Burd

Abstract

Energy-Efficient Processor System Design

by
Thomas David Burd

Doctor of Philosophy in Engineering-Electrical Engineering
and Computer Sciences

University of California, Berkeley

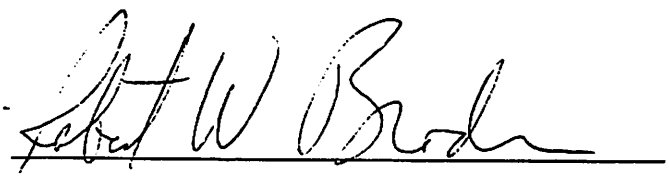
Professor Robert W. Brodersen, Chair

Motivated by the pervasive use of general-purpose processors in portable electronics devices, energy-efficient processor system design is presented as a critical enabler for smaller, more powerful, and longer-running devices. A decade of research has demonstrated that extremely energy-efficient ASIC and custom DSP design is achievable, but the energy-efficiency of general-purpose processors has severely lagged behind. Thus, despite only performing a small fraction of the computation in these portable devices, the processor system contributes a significant, if not dominant, fraction of the device's total energy consumption. This thesis introduces and demonstrates a top-down processor system design methodology for dramatically reducing energy consumption, while maintaining the desired level of performance.

By understanding the fundamental usage requirements of a processor in portable devices, combined with analytical models for energy consumption and performance, energy-efficiency metrics are derived. A key design technique derived from these metrics, dynamic voltage scaling, is then described for achieving the single-largest increase in energy-efficiency. The metrics are further utilized in developing an

overall energy-conscious design flow methodology, and more specifically, energy-efficient architectural and circuit design methodologies to additionally improve system energy-efficiency.

The design and measured results are reported on a prototype processor system, which successfully demonstrates the design techniques and methodologies presented in this thesis, and the potential improvement in processor system energy-efficiency. The system consists of four custom chips: a microprocessor, an SRAM, a voltage converter, and an I/O interface chip. On top of this system runs a real-time operating system, which executes software programs typically found in portable devices, to demonstrate a complete embedded processor system. This prototype system's energy-efficiency was quantified, and demonstrated to be more than order of magnitude higher than the most energy-efficient processor system available today.



Robert W. Brodersen, Chairman of Committee

To my wife,

Joyce Law

and my parents,

Mike and Lois Burd

Table of Contents

Acknowledgments

Through the course of this thesis, I have spent many years at Berkeley in which I have had the chance, and privilege, to interact with many brilliant people and outstanding engineers. This interaction has helped me develop as both an engineer, and an individual, and I have many people to thank for their help towards the culmination of this work. Including my undergraduate years, I have spent almost thirteen years at Berkeley, and what a long, strange trip it has been.

First and foremost, I would like to thank my wife, Joyce, for her many years of love and friendship. I don't know if I could have made it through graduate school without her companionship, her understanding, as well as her common sense. She has enriched my life in ways too numerous to count, and I am forever indebted to her.

I am also extremely grateful for my parents, Mike and Lois, for their love, support, and guidance, which was instrumental in all my life's accomplishments. I would have liked to have had my mom celebrate my graduation with me, and hope that even in the heavens above, she can be proud. Additional thanks go to my dad, who taught me over the years how to be a practical engineer. Finally, I won't be hearing that question anymore, "so when are you going to graduate?"

While I thank all my siblings for their friendship and love, my brother, Bob, deserves special thanks for the years of mentoring, the Opel, my first internship, my first consulting job, for providing a roof over my head in my time of need, as well as for occasionally being my banker. It is because of him that I originally came to Berkeley many years ago, and where I am in life today.

My desire to pursue a graduate degree did not develop until late in my undergraduate years. I would like to thank Professor Donald O. Pederson for providing

me with a valuable undergraduate research experience which initiated my desire to continue on with graduate research, and Dr. Tom Andrade for his valuable mentorship as my first engineering supervisor, who also encouraged me to pursue a Ph.D.

It has truly been an honor and pleasure to have had Professor Robert W. Brodersen as my graduate advisor, and I thank him for his years of support, advice, and encouragement. He has been critical in teaching me to be an independent thinker and to always look at the "big picture". I thank Bob for giving me the chance to work on the research that interested me most, despite contradicting his own research direction. Both Joyce and I will be eternally grateful for the deferred admission. Otherwise, we might be living in Boston now.

I would like to thank Professor Jan Rabaey for his advice and support over the years, especially towards the end, for providing the DARPA funding for my research. I thank Professor Randy Katz for chairing my quals committee, and for broadening my knowledge of mobile networking. Professor David Culler, who I was also honored to have sit on my quals committee, taught me much of what I know about computer architecture.

During my earlier years when I worked on the InfoPad research project, I was extremely fortunate to have worked with many outstanding graduate students. I first worked with Anantha Chandrakasan while still an undergraduate, and am grateful to him for convincing Bob to take me on as a graduate student, and for teaching me about low-power DSP design. I thank Andy Burstein for sharing with me his brilliant insight of circuit design, as well as his incredible cooking. I will always remember the day Andy's ball burst. I thank Sam Sheng for sharing with me his insight of circuit design, and for the endless hours he selflessly spent helping me with problems I was trying to solve. As cubemates, both Andy and Sam made the many hours I spent in 550H Cory pass more easily.

There are many others who I would like to thank for the learning experience we had while working on the InfoPad research project. In addition to Anantha and Andy, Shankar Narayanaswamy, Tom Truman, Kathy Lu, and Richard Edell were all critical in getting the first Infopad prototype built and operational, with whom I remember spending many long nights in 403 Cory. While I only worked alongside him for a short while, I would like to thank Mani Srivastava for all his help with dpp.

I thank both Trevor Pering and Tony Stratakos, who were integral to this research, for the pleasure I had working alongside them during this thesis. Because of their tremendous efforts, we will always be able to look back upon our ISSCC Best Paper award and be proud of the work we accomplished at Berkeley. I wish both of the them the best in their future endeavors, and will forever be grateful for their efforts. I would also like to thank several others who have helped me on this thesis at various points over the years: Peggy Laramie, Omid Rowhani, Vandana Prabhu, Patrick Chiang, Chris Chang, Kevin Camera, and Hayden So. I am particularly grateful to Peggy and Omid, who saved me from at least two more additional years of work, and who were critical in getting working first silicon.

I would also like to thank the many students in BJgroup that I have worked with over the years. It was truly an outstanding group of students covering a broad range of disciplines, from whom I learned many things. More importantly, I made many new friends with whom I had a great time at Berkeley.

Those that taught me many things about RF over the years include Bill Barringer, Dennis Yee, Lapoe Lynn, Kevin Stone, Arya Behzad, Craig Teuscher, Jennie Chen, Dave Sobel, Chinh Doan, Brian Limketkai, and Sayf Alalusi. I learned much of what I know about CAD from Paul Landman, Dave Lidsky, Lisa Guerra, Renu Mehra, Ole Bentz, and Marlene Wan. Other masochists who worked on large hardware projects, with whom I could console with, included Arthur Abnous, Varghese George, Alfred Yeung, and Ian O'Donnell. From Ingrid Verbaughwe I learned a lot about DSP. Other

people that I had the pleasure of working with on InfoPad include Jeff Gilbert, Heather Bowers, Roy Sutton, John Davis, Fred Burghardt, Rich Han, and Roger Doering. Additionally, I enjoyed working with many people at the BWRC, including Kostas Sarrigeorgidis, Henry Jen, Andy Klein, Chris Taylor, Dejan Markovic, Johan Vanderhaegen, Hui Zhang, Ada Poon, and Ning Zhang.

Over the years, I was fortunate to work with, and learn from, many excellent professors. Special thanks go to Professor Bora Nikolic for going over this thesis with a fine-tooth comb and providing his valuable comments. I thank Professor Paul Wright for also reading my thesis, and from whom I learned immensely about mechanical engineering issues in product design. From Professor Bernhard Boser, I learned extensively about analog circuits by teaching his 241 class through NTU many times, and likewise, Professor Paul Gray. I would also like to thank Pam Atkinson for doing such an excellent job of running NTU for all the semesters I taught classes.

I am very grateful for the many friends I made in 550 Cory outside of BJgroup for making it such a fun place to work. Chris Rudell, for his hilarious imitations and stories, the bike rides, the steaks, the many dinners out, and numerous other good times. Despite his love for polluting my air space, I've been fortunate to have such a great friend. Andy Abo, who is such a fun guy to party with, and who was a great host during my time in Japan. Sekhar Narayanaswami, my fellow Cowboy's fan and Cal fan, with whom I spent many hours rooting our teams on. Keith Onodera, who scared me the first time he whipped out his poker chip set. Jeff Weldon, for all his help on my golf game and clubs, and with whom I've enjoyed discussing the finer things in life. There are many others to which I owe thanks for their technical help, and with whom I enjoyed hanging out with: Srenik Mehta, Jeff Ou, George Chien, Adrian Isles, Carol Barrett, and Anna Ison, among others.

I am very appreciative for the spectacular staff support I have had, both in the department, and in the research group. The foremost person I need to thank is Tom

Boot, who always lent me a helping hand with any and all things administrative. I thank Ruth Gjerde for all her help navigating the grad office. Sue Mellers provided me well-needed help with my various test boards. Many thanks go to Kevin Zimmerman, for help with all things related to the computer network, and Brian Richards for about everything else technical. Elise Mills and Peggye Browne were a huge help with all my various purchase orders, and while I did not interact much with Deirdre Bauer in the office, she was always a fun person at the research retreats.

1 Introduction

The explosive proliferation of portable electronic devices has compelled energy-efficient VLSI and system design to provide longer battery run-times, and more powerful products that require ever-increasing computational complexity. In addition, the demand for low-cost and small form-factor devices has kept the available energy supply roughly constant by driving down battery size, despite advances in battery technology which have increased battery energy density. Thus, energy-efficient design must continuously provide more performance per watt.

Since the advent of the integrated circuit (IC), there have been micro-power ICs which have targeted ultra-low-power applications (e.g. watches) with power dissipation requirements in the micro-Watt range [vitt80]. However, these applications also had correspondingly low performance requirements, and the ICs were not directly applicable to emerging devices (e.g. cell phones, portable computers, etc.) with much higher performance demands.

For the last decade, researchers have made tremendous advancements in energy-efficient VLSI design for these devices, derived, in part, from the earlier micro-power research, and targeted towards the devices' digital signal processing (DSP). Initial work demonstrated how voltage minimization, architectural modification such as parallelism and pipelining, and low-power circuit design could reduce energy

consumption in low-power custom DSP application-specific ICs (ASICs) by more than 100x [chan92]. Later work demonstrated significant energy-efficiency improvement for a variety of signal-processing applications, including the custom ASICs in a portable multimedia terminal [chan94], a custom video decoder ASIC [tser96], and programmable DSP ICs [ueda93][shir96][lee97].

A common component in these portable devices is a general-purpose processor. Few devices are implemented with a full-custom VLSI solution, as a processor provides two key benefits: the ability to easily implement control functionality which does not map to custom hardware, and more importantly, the ability to upgrade and/or modify functionality, after implementation, due to its programmable nature. Although the processor may perform as little as 1% of the total device computation, advances in energy-efficient custom DSP implementation have made the processor power dissipation a dominant component in portable devices.

Since the advent of the first integrated CMOS microprocessor, the Intel 4004 in 1971, microprocessors were consistently designed with one goal in mind: performance. Processor power and silicon area had been relegated to secondary concern. The wide-spread emergence of portable devices has created a demand for more energy-efficient processors, but the industry trend has been to fabricate an older processor in a better process technology, operate it at a reduced supply voltage, and market it as a low-power processor. Process and voltage scaling does improve energy-efficiency, but not the improvement possible with a whole-scale processor redesign with energy consumption in mind from the outset.

While some processors have been touted as low-power, and have become quite prevalent in portable devices, they generally achieved this by delivering lower performance. Thus, they are low-power, but not necessarily energy-efficient. The StrongArm processor demonstrated what can be achieved by designing a processor with energy consumption in mind from the start [mont96]. It provided a five-fold increase in

energy-efficiency, as compared to other contemporary processors, which had otherwise only demonstrated incremental increases.

But even the StrongArm remains 100x-1000x less energy-efficient for basic computation (e.g. arithmetic, logical operations) than a custom ASIC implementation [zhan00]. This is in large part due to the overhead required by a general-purpose processor: fetching and decoding instructions, multiplexing instructions onto the same underlying hardware, and supporting superscalar and/or pipelined microarchitectures. However, there is still large room for improvement to further improve processor energy-efficiency, as will be demonstrated throughout this thesis.

1.1 The Performance-Energy Trade-off

The processor performance and energy consumption is shown in Figure 1.1 for some portable devices currently available. While notebook computer processors deliver tremendous amounts of performance, their high energy consumption requires a large battery to provide even a few hours of run-time. On the other hand, Palm-PCs and PDAs can deliver increasingly longer battery run-time due to their lower energy consumption,

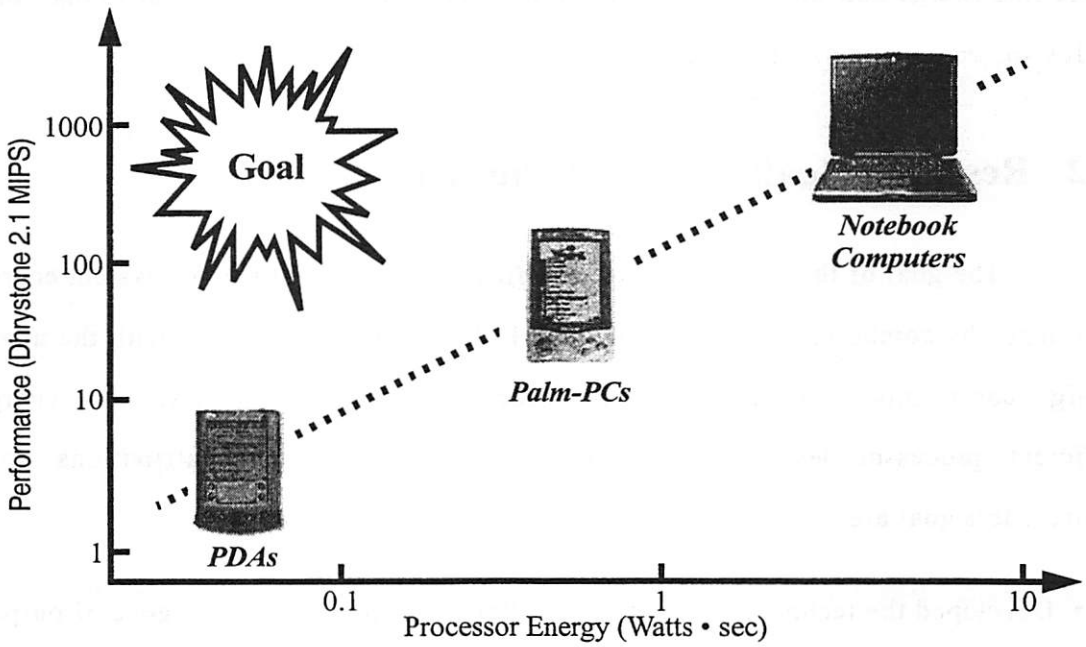


FIGURE 1.1 : Processor Performance vs. Energy Consumption.

but it comes at the expense of decreased performance.

In fact, there is a general performance-energy trade-off, as indicated by the dotted line, which occurs because many existing low-power design techniques sacrifice performance in order to achieve lower power. In DSP applications, parallelism is a common design technique to recover lost performance, while maintaining constant energy consumption. However, for a general-purpose microprocessor, parallelism has diminishing returns on increasing performance, and comes at the expense of exponentially increasing energy consumption.

The current philosophy has been to rely on process technologies improvements to shift the trend line up, which it does by approximately 2x per process generation. In breaking with this philosophy, the StrongArm processor pushed the trend line up 5x, and demonstrated that an energy-efficient processor design could yield as much improvement as two or three process generations.

But this still falls dramatically short of the ideal goal of a processor that could deliver performance approaching that of a notebook computer, while maintaining PDA-like energy consumption, and providing an energy-efficiency similar to that found in low-power, custom ASIC designs.

1.2 Research Goals and Contributions

The goal of this research is to significantly improve processor system energy-efficiency by combining the lessons learned in low-power DSP design with the unique design constraints of a general-purpose processor to develop a new, more energy-efficient, processor design methodology. Several key research contributions which address this goal are:

- Developed the technique of Dynamic Voltage Scaling (DVS) for a general-purpose microprocessor to dynamically vary the processor's supply voltage and clock

frequency, under operating system control. This allows the processor to provide high performance when required, while minimizing energy consumption during the remaining low-performance periods of time. This technique is most significant because it eliminates the energy-performance trade-off of more traditional low-power design techniques.

- Developed an energy-conscious design flow which enables energy consumption optimization at all levels of the design flow, including the high-level C behavioral simulator, where optimizations can have the biggest impact on energy-efficiency. The new flow also eliminates the extra complexity added by DVS to a more traditional design flow.
- Developed an energy-efficient architectural design methodology for all aspects of a processor system, including system-level optimizations, as well as optimizations targeted for the processor core and cache system.
- Developed an energy-efficient circuit design methodology for all aspects of digital circuit design for processors, while meeting the circuit design constraints imposed by DVS.
- Demonstrated the above concepts by implementing a prototype processor system, consisting of four custom chips in a 0.6μm CMOS process technology, that can operate over the range of 1.2-3.8V, 5-80MHz, and 0.54-5.6 mW/MIP. Through DVS, the system can deliver a peak performance of 85 Dhrystone 2.1 MIPS, with an average power dissipation as low as 3.24mW. This yields as much as 26,000 MIPS/W, which is more than 10x than the most energy-efficient microprocessor currently available.

1.3 Organization

Chapter 2 presents a usage model for processors found in portable electronic devices in order to qualitatively identify the critical design optimizations for processor

performance and energy consumption. Analytical CMOS circuit models are then presented, from which three metrics are derived to quantify energy-efficiency. Four key energy-efficient design principles are presented to demonstrate the application of these metrics.

Dynamic voltage scaling (DVS), a technique to dynamically vary a processor's performance and energy consumption, is presented in Chapter 3. After demonstrating the potential energy-efficiency improvement of DVS, a voltage converter is described which generates the variable voltage and clock frequency. The design constraints placed upon the processor's circuits are then examined. DVS requires operating system support via the voltage scheduler, which is presented, followed by new benchmark programs used to quantify the energy-efficiency improvement of DVS.

An energy-conscious design flow methodology is described in Chapter 4, which constantly evaluates not only performance, but energy consumption as well, at all levels of the design hierarchy. A majority of the design cycle in modern complex processor designs is spent on validating functionality, a problem which is exacerbated by DVS. The remainder of the chapter describes four parts of the design flow that were developed to aid and speed-up the design of a DVS processor system: clocking methodology, power distribution methodology, functional verification, and timing verification.

Chapter 5 presents a top-down energy-efficient architectural design methodology. First, system-level architectural design issues are discussed, followed by a more in-depth analysis of the processor core and the cache system.

An energy-efficient circuit design methodology is described in Chapter 6. General circuit design techniques are discussed, including choosing logic styles, transistor sizing, clock-gating, optimizing interconnect, and layout considerations for both standard and datapath cell libraries. Memory design, which has additional

constraints placed upon is by DVS, is presented next. Low-swing bus transceivers are then described, which can be used to significantly reduce energy consumption for on-chip busses, and even more significantly, for inter-chip busses.

A prototype processor system, consisting of four custom chips, is presented in Chapter 7. This system successfully demonstrates the energy-efficiency improvement due to DVS (5-10x), as well as the energy-efficiency improvement due to the previously described energy-efficient design methodology (2-3x). By radically re-evaluating the design of a processor, its energy efficiency has been improved by more than a factor of 10x, as compared to currently-available commercial processors, which even have the benefit of much better process technologies.

Chapter 8 provides concluding remarks and suggestions for future research directions.

2 Energy Efficient Design

To effectively optimize the energy efficiency of a processor system, it is critical to first understand the computational demands placed upon it. Based upon relatively simple CMOS circuit models suitable for deep sub-micron process technologies, three energy-efficiency metrics will be derived. Finally, some key energy-efficient design principles will be discussed to demonstrate the application of these metrics.

2.1 Processor Usage Model

Understanding a processor's usage pattern is essential to its optimization. Processor utilization can be evaluated in terms of the amount of processing required and the allowable latency for the processing to complete. These two parameters can be merged into a single measure, which is Throughput, or *T*. It is defined as the number of operations that can be performed in a given time:

$$\text{Throughput} \equiv T = \frac{Operations}{Second} \qquad \text{(EQ 2.1)}$$

Operations are defined as the basic unit of computation and can be as fine-grained as instructions or more coarse-grained as programs. This leads to measures of throughput of MIPS (instructions/sec) and SPECint95 (programs/sec) [spec94] which compare the throughput on implementations of the same instruction set architecture

(ISA), or different ISAs, respectively.

2.1.1 Processor Operation

The desired throughput of various software processes executing on a processor are shown in Figure 2.1. The example processor usage pattern shows that the desired throughput varies over time, and the type of computation falls into one of three categories.

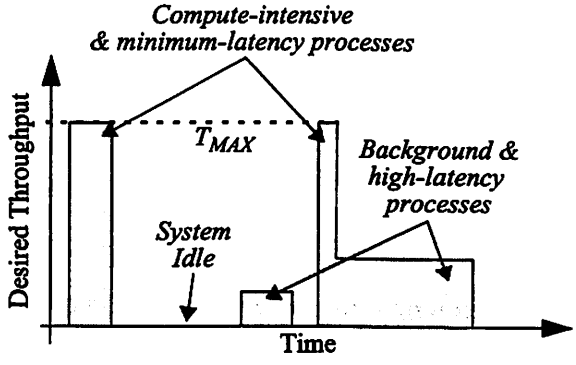


FIGURE 2.1 : Processor Utilization.

Compute-intensive and minimum-latency processes desire maximum performance, which is limited by the peak throughput of the processor, T_{MAX}. Any increase in T_{MAX} that the hardware can provide will readily be used by these processes to reduce their latency. Examples of these processes include spreadsheet updates, document spell checks, video decoding, and scientific computation.

Background and high-latency processes require just a fraction of the full throughput of the processor. There is no intrinsic benefit to exceeding the real-time latency requirements of the process since the user will not realize any noticeable improvement. Examples of these processes include video screen updates, data entry, audio/video codecs, and low-bandwidth I/O data transfers.

The third category of computation is system idle, which has zero desired throughput. Ideally, the processor should consume zero power in this mode and therefore be inconsequential. However, in any practical implementation, this is not the

case. Hence, as will be discussed in Section 5.2.5, optimizing this mode of operation requires special attention.

These three modes are found in most single-user processor systems, from personal digital assistants (PDAs), to notebook computers, to powerful desktop machines. This model does not apply to systems implementing a fixed-rate DSP algorithm; these systems operate either in the fixed-latency or idle modes of operation which are much better suited to be implemented in a custom DSP ASIC [chan95]. In multi-user mainframe computers, where the processor is constantly in use, this usage model also does not hold true. For these machines, the processor essentially spends the entire time in the compute-intensive mode of operation.

2.1.2 What Should be Optimized?

Any increase in processor speed can be readily exploited by compute-intensive and minimum-latency processes. In contrast, the background and high-latency processes do not benefit from any increase in processor speed above and beyond their average desired throughput since the extra throughput cannot be utilized. Thus, peak throughput is the parameter to be maximized since the average throughput is determined by the user and/or operating environment.

The run-time of a portable system is typically constrained by battery life. Simply increasing the battery capacity is not sufficient because the battery has become a significant fraction of the total device volume and weight [culb94][iked95][kuni95]. Thus, it has become imperative to minimize the load on the battery, while simultaneously increasing the speed of computation to handle ever more demanding tasks. Even for wired desktop machines, the drive towards "green" computers are making energy-efficient design a priority. Therefore, the computation per battery-life/ Watt-hour should be maximized, or equivalently, the average energy consumed per operation should be minimized.

Due to the high cost of heat removal, it has also become important to minimize peak energy consumed per operation (i.e. power dissipation), mainly in high-end computing machines and notebook computers. However, the focus of this work is on energy-efficient computing, so the parameter that this work focuses on is average energy consumption.

2.1.3 InfoPad: A Case Study in Energy Efficient Computing

The InfoPad is a wireless, multimedia terminal that fits a compact, low-power package in which much of the processing has been moved onto the backbone network [chan94]. An RF modem sends/receives data to/from five I/O ports: video output, text/ graphics output, pen input, audio input, and audio output. Each I/O port consists of specialized digital ICs, and the associated I/O device (e.g. LCD, speaker, etc.). In addition, there is an embedded processor subsystem used for data flow and network control. InfoPad provides an interesting case study because it contains large amounts of data processing and control processing, which require different optimizations for energy efficiency.

The specialized ICs include a video decompression chip-set which decodes 128×240 pixel frames in real-time, at 30 frames per second. The collection of four chips takes in vector quantized data and outputs analog RGB directly to the LCD and dissipates less than 2mW. Implementing the same decompression in a general purpose processor would require a throughput of around 10 MIPS with hand-optimized code. A processor subsystem designed with the best available parts in an 1.2μm equivalent process technology would dissipate at least 200mW. This provides a prime example of how dedicated architectures can radically exploit the inherent parallelism of signal processing functions to achieve orders of magnitude reduction of power dissipation over equivalent general-purpose processor-based systems.

The control processing, which has little parallelism to exploit, is much better

suited towards a general purpose processor. An embedded processor system was designed around the ARM60 processor [gec94], which combined with SRAM and external glue logic dissipates 1.2W, while delivering a peak throughput of 10 MIPS. It is this discrepancy of almost three orders of magnitude in power dissipation that leads to this work's objective of substantially reducing the processor system's energy consumption.

2.1.4 The System Perspective

In an embedded processor system such as that found in InfoPad, there are a number of digital ICs external to the processor chip required for a functional system: main memory, clock oscillator, I/O interface(s), and system control logic (e.g., PLD). Integrated solutions have been developed for embedded applications that move the system control logic, the oscillator, and even the I/O interface(s) onto the processor chip leaving only the main memory external such as the SA-1100 processor [dec98].

Figure 2.2 shows a schematic of the InfoPad processor subsystem, which contains the essential system components described above. Interestingly, the processor does not dominate the system's power dissipation; rather, it is the SRAM memory which dissipates half the power. For aggressive energy-efficient design, it is imperative to optimize the entire system and not just a single component; optimizing just the processor in the InfoPad system can yield at most a 10% reduction in power.

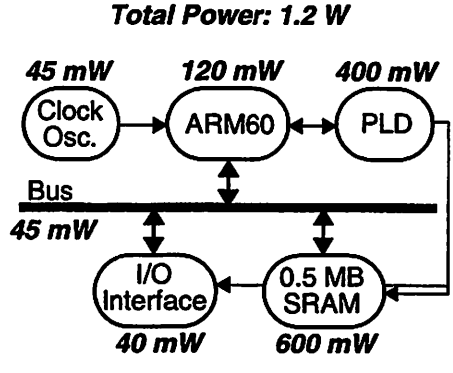


FIGURE 2.2 : InfoPad Processor Subsystem.

High-level processor and system simulation is generally used to verify the functionality of an implementation and find potential performance bottlenecks. Unfortunately, such high-level simulation tools do not exist for energy consumption, which forces simulations to extract energy consumption to be delayed until the design has reached the logic design level. At this time, it is very expensive to make significant changes, because it is difficult to make system optimizations for energy consumption through whole-scale redesign or repartitioning.

It is important to understand how design optimizations in one part of a system may have detrimental effects elsewhere. A simple example is the relative effect of a processor's on-chip cache on the external memory system. Because smaller memories have lower energy consumption, the designer may try to minimize the on-chip cache size to minimize the energy consumption of the processor at the expense of a small decrease in throughput (due to increased miss rates of the cache). However, the increased miss rates affect not only the performance, but may increase the system energy consumption as well because high-energy main memory accesses are now made more frequently. So, even though the processor's energy consumption was decreased, the total system's energy consumption has increased.

2.2 CMOS Circuit Models

CMOS has become the predominant process technology for digital circuits. Circuit delays and power dissipation for CMOS circuits can be accurately modeled with simple equations, even for complex processor circuits. These models, along with knowledge about the system architecture, can be used to derive analytical models for energy consumed per operation and peak throughput.

These models will be presented in this section and then used in Section 2.3 to derive metrics that quantify energy efficiency. With these metrics, the circuit and system design can be analytically optimized for maximum energy efficiency.

2.2.1 Power Dissipation

There are four main sources of power dissipation: dynamic switching power due to the charging and discharging circuit capacitances, short-circuit current power due to finite signal rise and fall times, leakage current power from reverse-biased diodes and subthreshold conduction, and static biasing power found in some types of logic styles (i.e. pseudo-NMOS).

Typically, the power dissipation is dominated by the dynamic switching power. However, it is important to understand the other components as they can have a significant contribution to the total power dissipation in poorly-designed integrated circuits.

2. 2. 1. 1 Dynamic Switching Power

For every low-to-high output transition in a digital CMOS gate, the capacitance on the output node, C_L, incurs a voltage change ΔV, drawing an energy of $C_L \cdot \Delta V \cdot V_{DD}$ Joules from the supply voltage, V_{DD} [chan95]. A high-to-low transition dissipates the energy stored on the capacitor into the NMOS device(s), pulling the output low. The power dissipation is just the product of the energy consumed per transition and the rate at which low-to-high transitions occur, $F_{0\rightarrow 1}$.

For the simple inverter gate shown in Figure 2.3, ΔV is equal to V_{DD}, so the power drawn from the supply is:

$$Power_{INVERTER} = C_L \cdot V_{DD}^2 \cdot F_{0\rightarrow 1} \qquad \text{(EQ 2.2)}$$

This simple equation holds for more complex gates, and other logic styles as well, given a periodic input. In static logic design, the output only transitions on an input transition, while in dynamic logic, the output is precharged during half the clock cycle, which may force a transition, and a transition can also occur in the other half-cycle, depending upon the input values. In both cases, the power dissipated during switching is

proportional to the capacitive load; however, they have different transition frequencies.

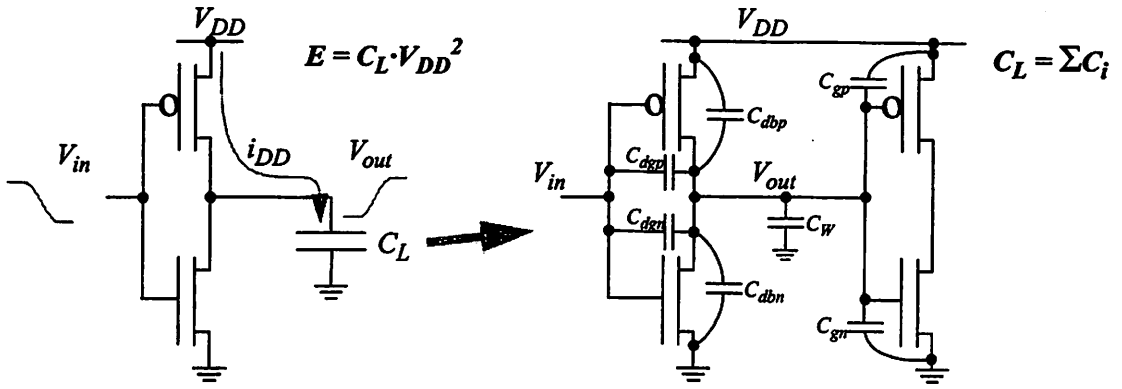


FIGURE 2.3 : Dynamic Switching Power Dissipation; Sources of Capacitance.

The basic capacitor elements of C_L, shown in Figure 2.3, consist of the gate capacitance of subsequent inputs attached to the inverter output (C_{gp}, C_{gn}), interconnect capacitance (C_W), and the diffusion capacitance on the drains of the inverter transistors (C_{dbp}, C_{dbn}, C_{dgp}, C_{dgn}) [raba96].

Usually, the value of $F_{0\to1}$ is difficult to quantify since it is typically not periodic, and is strongly correlated with the input test vectors. Without doing a transistor-level circuit simulation, the best way to calculate $F_{0\to1}$ is to either perform statistical analysis on the circuit [land93], or use a high-level behavioral model with benchmark software to determine a mean value. Since most digital CMOS circuits are synchronous with a clock frequency, f_{CLK}, an activity factor, $0 < \alpha < 1$, is used to denote the average fraction of clock cycles in which a low-to-high transition occurs, such that $F_{0\to1} = \alpha \cdot f_{CLK}$.

For an integrated circuit with N nodes, the total dynamic switching power is:

$$Power_{DYNAMIC} = V_{DD} \cdot f_{CLK} \cdot \sum_{i=1}^{N} \alpha_i \cdot C_{Li} \cdot \Delta V_i \qquad \text{(EQ 2.3)}$$

Aside from memory bit-lines and low-swing logic, most nodes swing $\Delta V = V_{DD}$, as was the case for the simple inverter, so that the power equation can be simplified to:

$$Power_{DYNAMIC} \cong V_{DD}^2 \cdot f_{CLK} \cdot C_{EFF} \qquad \text{(EQ 2.4)}$$

where the effective switched capacitance, C_{EFF}, is commonly expressed as the product of the physical capacitance C_L and the activity weighting factor α, each averaged over the N nodes.

2. 2. 1. 2 Short-Circuit Current Power

Short-circuit currents occur when the output of a gate is transitioning while the input is still in mid-transition. This generally occurs when the rise/fall time at the input is larger than the output rise/fall time. For the ideal case of a step input, the transistors change state immediately, one turning on, one turning off. There is no conductive path from the supply to ground. For actual circuits, however, the input signal will have a finite rise/fall time. While the conditions $V_{Tn} < V_{in} < V_{DD} - |V_{Tp}|$ and $0 < V_{out} < V_{DD}$ hold for the input/output voltages, there will be a conductive path open because both devices are on.

The longer the input rise/fall time, the longer the short-circuit current will continue to flow, and the average short-circuit current increases. Figure 2.4 plots the increase in energy consumption due to short-circuit current versus the ratio of input rise/fall time (t_{in}) to output rise/fall time (t_{out}) for a static CMOS inverter. The $\Delta E/E_{(tin=0)}$ increases dramatically with increasing input rise/fall time. To minimize the

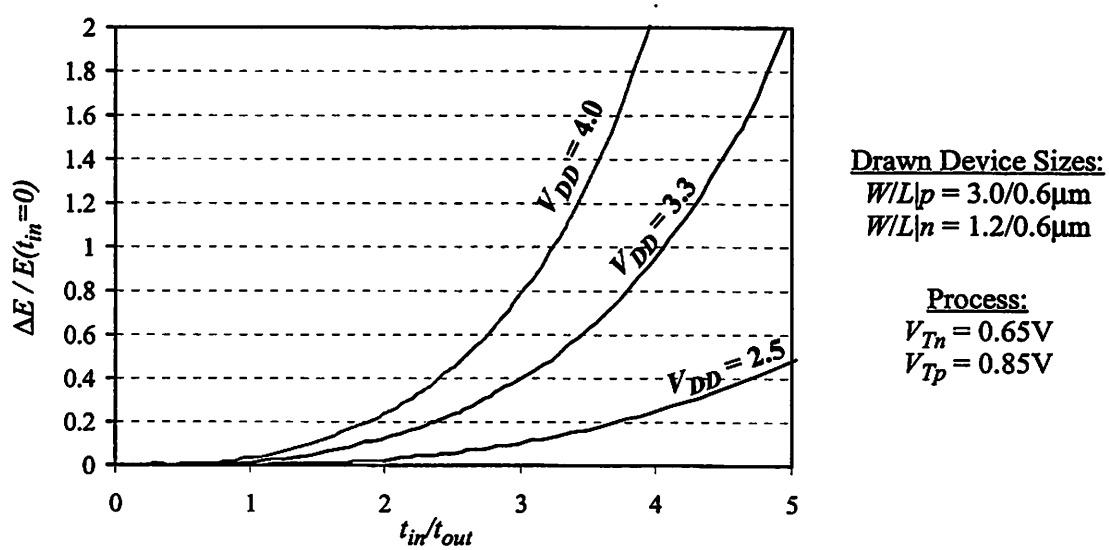


FIGURE 2.4 : Short-circuit Energy Consumption vs. Input Rise/fall Time.

total average short-circuit current power, it is desirable to have equal input and output rise/fall times, since the input rise/fall time of one gate is the output rise/fall time of another gate.

The average short-circuit current is roughly independent of device size for a fixed load capacitance, since even though the peak magnitude of the current scales with device width, the rise/fall time scales inversely with device width such that the average current is approximately the same. The fraction of power dissipation due to short-circuit current scales with V_{DD}. However, when the supply is lowered to below the sum of the thresholds of the transistors, $V_{DD} < V_{Tn} + |V_{Tp}|$, short-circuit currents will be eliminated because both devices cannot be on at the same time.

For well-designed ICs, the short-circuit power dissipation can be limited to 5-10% of the total dynamic power [veen84]. This is achieved by maintaining a bounded ratio on rise/fall times through a transistor-width sizing methodology, discussed further in Section 6. 1. 2. 1, so that:

$$Power_{SHORT} = \delta_{SC} \cdot Power_{DYNAMIC} \qquad \text{(EQ 2.5)}$$

where δ_{SC} is the ratio of short-circuit to dynamic power dissipation.

2. 2. 1. 3 Leakage Current Power

There are two types of leakage currents: reverse-bias diode leakage, and sub-threshold leakage through the channel of an "off" device. The magnitude of these currents is set predominantly by the processing technology and total number of transistors.

Diode leakage occurs when one transistor is turned off, and another active transistor charges up, or down, the drain with respect to the former's bulk potential. For a static CMOS inverter, shown in cross-section in Figure 2.5, with a low input voltage, the output voltage will be high because the PMOS transistor is on. The NMOS transistor

will be turned off, but its bulk-to-drain voltage will be equal to the supply voltage, $-V_{DD}$. The resulting diode leakage current will be approximately $I_{LD} = A_D \cdot J_{SD}$, where A_D is the area of the drain diffusion, and J_{SD} is the leakage current density of the diffusion, set by the technology. Since the diode reaches maximum reverse-bias current for relatively small reverse-bias potential (< 100mV), the leakage current is roughly independent of supply voltage.

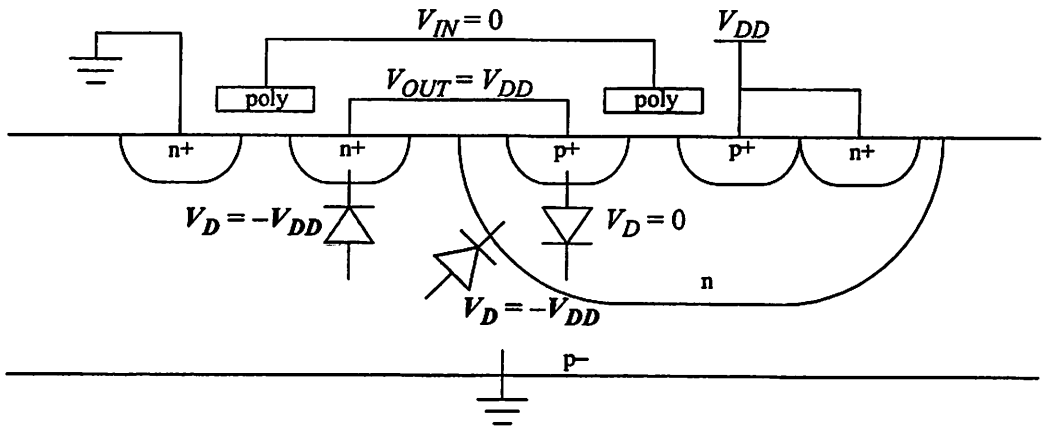


FIGURE 2.5 : Reverse-biased Diodes in CMOS Inverter.

In an nwell process, such as that depicted in Figure 2.5, the nwell-substrate reverse-biased diode also has leakage current. Since a diode's leakage current is primarily determined by the more lightly doped side of the junction, which is the p– substrate, the leakage current density is similar to that of the NMOS drain-substrate diode [mull86]. Because the well area, A_W, is an order of magnitude larger than the diffusion area, this current will dominate reverse-biased diode leakage in an n-well process. The current is $I_{LW} = A_W \cdot J_{SW}$, where J_{SW} is the leakage current density of the well, also set by the technology.

For the MOSIS 0.6μm process, $J_{SD} \approx 100\text{nA/m}^2$ and $J_{SW} \approx 100\text{nA/m}$ (at 25° C). The leakage current density is temperature sensitive, so J_S can increase dramatically at higher temperatures. Since the well-diode leakage dominates diffusion-diode leakage, the leakage current can be estimated from the size of the die. For a large 200mm^2 chip, approximately one-half the area is nwell, such that the total diode

leakage is on the order of 10pA.

Subthreshold leakage occurs under similar conditions as the diode leakage. In the inverter described above, the NMOS was turned off, but even for $V_{GS} = 0V$, there is still current flowing in the channel due to the V_{DS} potential of V_{DD}. The I_D vs. V_{GS} characteristic, as shown in Figure 2.6, has an exponential relation in the subthreshold region ($V_{GS} < |V_T|$).

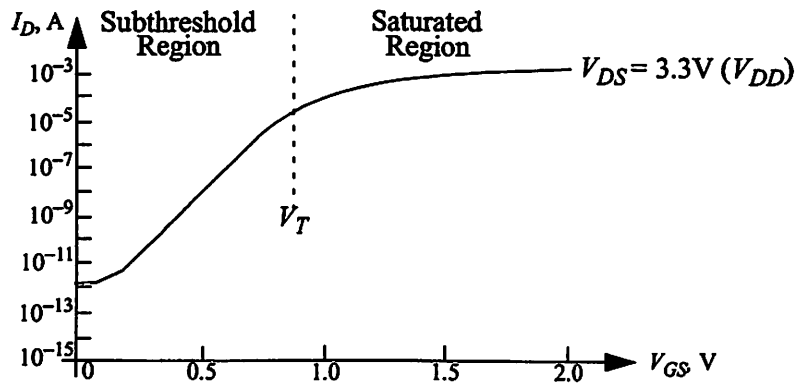


FIGURE 2.6 : I_D vs. V_{GS} for MOSFET in Subthreshold Region.

The magnitude of the subthreshold current is both a function of process, device sizing (W/L), and supply voltage [sze81]. The process parameter that predominantly affects the current value is V_T. Reducing V_T exponentially increases the subthreshold current, which to first order, is proportional to V_{DS}, or equivalently, V_{DD}.

For a V_T of 0.8V, the current magnitude for a single device is on the order of 1pA. Approximately one out of every two transistors has the necessary bias conditions for subthreshold leakage. For a 2 million transistor chip, its total subtreshold current would be on the order of 1μA, which dwarfs the reverse-diode leakage current.

The combination of diode-leakage current and subtreshold current for the 2 million transistor chip is approximately 1μA, which at a supply voltage of 3.3V, is below 10μW. This is insignificant to the dynamic switching power while the processor is operating. This power is only important in setting the lower threshold of achievable

power dissipation while the processor is idling. Hence, this power component will be ignored except when discussing idle energy consumption.

However, as process technology continues to advance, the maximum operating voltage decreases, and reductions in V_T are required to maintain a reasonable gate-drive voltage. This is particularly true in processes targeted towards high-performance ICs. For example, a 0.35μm process with $V_T = 0.35V$ has a leakage current on the order of 10nA per device [mont96][de99]. This yields 10mA of leakage current for the same 2 million transistor chip, and may become a significant fraction of the power dissipation in a very low-power chip. There are several design techniques to reduce this leakage current, such that it is once again only critical when considering idle energy consumption, including selectively increasing channel lengths, dual V_T devices [muto95], and dynamically varying V_T [kuro96].

2. 2. 1. 4 Static Biasing Power

While static bias currents are usually avoided in CMOS circuits, occasionally, they may prove to be beneficial. A typical application is for a large complex gate that cannot be implemented with dynamic logic due to asynchronous timing constraints.

Figure 2.7 contains an example gate; it is a wide AND-OR-Invert gate with asynchronous inputs. To implement this in full static CMOS would require several

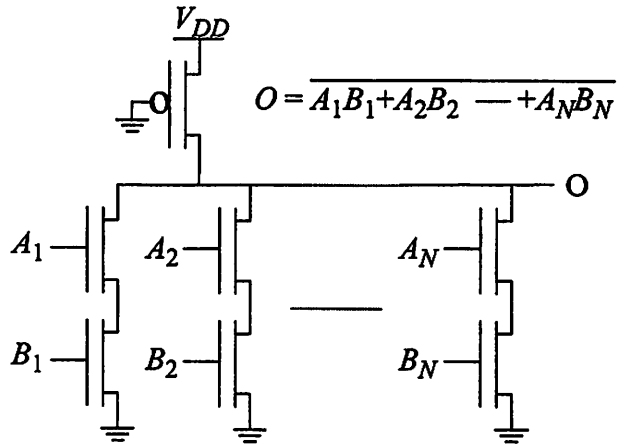


FIGURE 2.7 : Implementing Complex Logic with Static Biasing (pseudo-NMOS)

times the area to implement the stacked PMOS transistors. The extra PMOS transistors would also increase the capacitance on the input nodes, loading down the previous gates. However, by synchronizing the inputs through architectural design, which can usually be accomplished, and then implementing the complex gates with dynamic logic, this power component can be made negligible.

2. 2. 1. 5 Combined Power Model

With the assumption that no static biasing is present, the total power dissipation is just the summation of the remaining three individual components:

$$Power = Power_{DYNAMIC} + Power_{SHORT} + Power_{LEAKAGE} \quad \text{(EQ 2.6)}$$

where the $Power_{LEAKAGE}$ component is on the order of 10-100μW.

$$Power = (1 + \delta_{SC}) \cdot V_{DD}^2 \cdot f_{CLK} \cdot C_{EFF} + Power_{LEAKAGE} \approx V_{DD}^2 \cdot f_{CLK} \cdot C_{EFF} \quad \text{(EQ 2.7)}$$

To simplify the following analyses, the assumptions that $(1+\delta_{SC}) \approx 1$, and that $Power_{LEAKAGE}$ can be ignored except during processor idle will be made, so that the total chip power dissipation is approximately equal to just the dynamic switching power component.

2.2.2 Energy/Operation

A common measure of energy consumption is the power-delay product (PDP) [chan92]. This delay is often defined as the critical path delay, so PDP is equivalent to the energy consumed per clock cycle ($Power / f_{CLK}$). However, the measure of interest is the energy consumed per operation which can be derived by dividing the PDP by the operations per clock cycle. The energy consumed per operation can now be expressed as a function of effective switched capacitance, supply voltage, and operations per clock cycle:

$$\frac{Energy}{Operation} \cong \frac{V_{DD}^2 \cdot C_{EFF}}{Operations / Clock\ Cycle} \quad \text{(EQ 2.8)}$$

2.2.3 Circuit Delay

To fully utilize its hardware, a digital circuit should be operated at the maximum possible frequency. This maximum frequency is just the inverse of the delay of the processor's critical path which is proportional to the delay of a single CMOS gate

The delay for a CMOS gate, which is defined as the time required for the output to transition to 50% of the voltage swing, V_{DD}, can be approximated as [raba96]:

$$Delay \cong \frac{C_L}{I_{AVE}} \cdot \Delta V_{0\% \rightarrow 50\%} = \frac{C_L}{I_{AVE}} \cdot \frac{V_{DD}}{2} \qquad \text{(EQ 2.9)}$$

I_{AVE} is the average device current during the transition. For sub-micron MOS devices in velocity saturation, the device current, I_D, is [toh88]:

$$I_D = v_{SAT} \cdot C_{OX} \cdot W \cdot (V_{DD} - V_T - V_{Dsat}) \qquad V_{DS} \geq V_{Dsat} \qquad \text{(EQ 2.10)}$$

with the assumption of a fast input signal transition so that the device's gate-source voltage is V_{DD}. The term v_{SAT} is the maximum carrier velocity, which is a constant of bulk silicon [mull86]. C_{OX} is the gate capacitance, W is the device width, and V_T is the device threshold. V_{Dsat} is the value above which the current is independent of the drain-source voltage, V_{DS}, which in velocity saturation is [toh88]:

$$V_{Dsat} = \left(\frac{E_C \cdot L_e}{V_{DD} - V_T + E_C \cdot L_e}\right)(V_{DD} - V_T) \qquad \text{(EQ 2.11)}$$

where L_e is the effective electrical channel length, while E_C is the longitudinal electrical field at which the carriers are considered at v_{SAT}, and is a fundamental constant of silicon. E_C is approximately 1.5×10^6 V/m [pier96], so with the reasonable approximation that $V_T \sim E_C L_e$ (e.g. if $L_e = 0.5\mu m$, $V_T = 0.75V$):

$$V_{Dsat} \approx \left(\frac{V_T}{V_{DD}}\right)(V_{DD} - V_T) \qquad \text{(EQ 2.12)}$$

The device remains in saturation during the output transition (defined as a 50% change in output voltage), since V_{Dsat} is well below $\frac{1}{2}V_{DD}$ for the entire range of V_{DD},

and the current is approximately constant such that $I_{AVE} = I_D$. Combining equations 2.12, 2.10, and 2.9 yields:

$$Delay \cong \frac{C_L \cdot V_{DD}^2}{k_v \cdot W \cdot (V_{DD} - V_T)^2} \qquad k_v = 2 \cdot v_{SAT} \cdot C_{OX} \qquad \text{(EQ 2.13)}$$

where k_V contains the technology dependence. For our 0.6μm process ($L_e = 0.4$, $V_T = 0.75$V), this approximation gives less than 10% error in comparison to a SPICE-simulated delay using a BSIM3 device model [huan93].

2.2.4 Throughput

Throughput was previously defined as the number of operations that can be performed in a given time. When clock rate is set to be the inverse of the critical path delay, the throughput is equal to the amount of computational concurrency (i.e. operations completed per clock cycle) divided by this delay:

$$T = \frac{Operations}{Second} = \frac{Operations\ per\ Clock\ Cycle}{Critical\ Path\ Delay} \qquad \text{(EQ 2.14)}$$

The critical path delay can be related back to the previous delay model by summing up the delay over all M gates in the critical path:

$$Critical\ Path\ Delay \cong \frac{V_{DD}^2}{k_v \cdot (V_{DD} - V_T)^2} \cdot \sum_{i=1}^{M} \frac{C_{Li}}{W_i} \qquad \text{(EQ 2.15)}$$

Making the approximation that all gate delays are equal, Equation 2.15 can be simplified if N_{gates} is used to indicate the length of the critical path (i.e. number of gates), and average values for C_L and W are used. Throughput can now be expressed as a function of a technology parameter, supply voltage, critical path length, and operations per clock cycle:

$$T \cong \frac{k_v \cdot W \cdot (V_{DD} - V_T)^2}{N_{gates} \cdot C_L \cdot V_{DD}^2} \cdot \frac{Operations}{Clock\ Cycle} \qquad \text{(EQ 2.16)}$$

Typical units for operations per clock cycle are MIPS/Mhz, and SPECint95/MHz when operations are respectively defined as instructions and benchmark programs.

2.3 Energy Efficiency Metrics

While the energy consumed per operation should always be minimized, no single metric quantifies energy efficiency for all digital systems. The metric is dependent on the system's throughput constraint. There are three main modes of computation: fixed throughput, maximum throughput, and burst throughput. Each of these modes has a clearly defined metric for measuring energy efficiency, as detailed in the following three sections. While single-user systems typically operate in the burst throughput mode, the other two modes are equally important since they are degenerate forms of the burst throughput mode in which the system may operate.

2.3.1 Fixed Throughput Mode

Many real-time systems require a fixed number of operations per second. Any excess throughput cannot be utilized, and therefore needlessly consumes energy. Systems with this characteristic will be defined as operating in the fixed throughput mode of computation, and they are typically found in digital signal processing applications in which the required throughput is set by a fixed-rate incoming or outgoing real-time signal (e.g., speech, audio, video).

$$Energy\ Efficiency|_{FIX} = \frac{Power}{Throughput} = \frac{Energy}{Operation} \qquad \text{(EQ 2.17)}$$

Previous work has shown that the metric of energy efficiency in Equation 2.17 is valid for the fixed throughput mode of computation [chan92]. A lower value implies a more energy-efficient solution. If a design can be made twice as energy efficient (i.e. reduce the energy/operation by a factor of two), then its sustainable battery life has been doubled, and since the throughput is constant, its power dissipation has been halved. For the fixed-throughput mode, minimizing the power dissipation is equivalent to minimizing the energy/operation.

Using the energy model in Section 2.2.2, the metric can be expressed as:

$$Energy\ Efficiency|_{FIX} = \frac{V_{DD}^2 \cdot C_{EFF}}{Operations\ /\ Clock\ Cycle} \quad \text{(EQ 2.18)}$$

The primary way to improve energy efficiency is to reduce supply voltage while maintaining the throughput constraint, which yields a quadratic improvement in energy efficiency. Additionally, reducing the effective switched capacitance will also improve efficiency. Optimizing the energy efficiency of this mode of computation has been the focus of much previous work, which has yielded a variety of low-power design techniques that provide significant efficiency improvements [chan95].

2.3.2 Maximum Throughput Mode

In most multi-user systems, primarily networked workstations and mainframes, the processor is continuously running. The faster the processor can perform computation, the better, yielding the defining characteristic of the maximum throughput mode of computation. Thus, this mode's metric of energy efficiency must balance the need for low energy/operation and high throughput, which is accomplished through the use of the Energy-to-Throughput Ratio, or *ETR*:

$$Energy\ Efficiency|_{MAX} = ETR = \frac{E_{MAX}}{T_{MAX}} = \frac{Power}{Throughput^2} \quad \text{(EQ 2.19)}$$

where E_{MAX} is the energy/operation, or equivalently power/throughput, and T_{MAX} is the throughput in this mode.

A lower *ETR* indicates lower energy/operation for equal throughput, or equivalently, indicates greater throughput for the same amount of energy/operation, satisfying the need to equally optimize throughput and energy/operation. Thus, a lower *ETR* represents a more energy-efficient solution.

The Energy-Delay Product (EDP) is a similar metric [horo94], but does not include the effects of architectural parallelism when the delay is taken to be the critical

path delay. For example, two processors may consume the same energy/operation and operate at the same clock frequency, but one processor can complete two operations per cycle, while the other processor can only complete one operation per cycle. Although the EDP for the two processors is the same, indicating that they have equivalent energy efficiency, the *ETR* for the first processor is one-half the *ETR* for the second processor, correctly indicating that the processor which can complete two operations per clock cycle is actually twice as energy efficient.

Throughput and energy/operation can be scaled with supply voltage, as shown in Figure 2.8 (the data for Figures 2.8-2.10 is derived from Equations 2.8 and 2.16, which models sub-micron CMOS processes); but, unfortunately, they do not scale proportionally. So while throughput and energy/operation can be varied by well over an order of magnitude to cover a wide dynamic range of operating points, the *ETR* is not constant for different values of supply voltage.

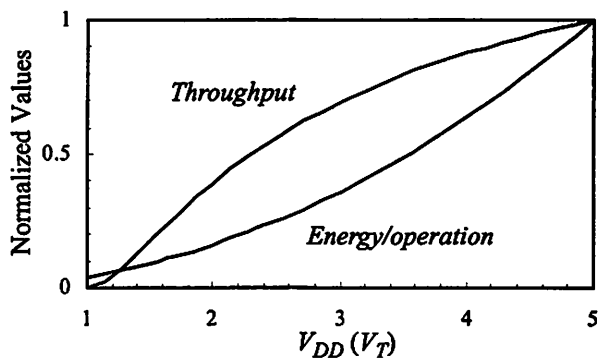


FIGURE 2.8 : Energy/operation, Throughput

As shown in Figure 2.9, V_{DD} can be adjusted by a factor of 2.5 (1.4-3.5V_T) and the *ETR* only varies within 50% of the minimum at $2V_T$. However, outside this range, the *ETR* rapidly increases. Clearly, for supply voltages greater than $4V_T$ there is a rapid degradation in energy efficiency, as well as for supply voltages that approach the device threshold voltage. But, since both throughput and energy/operation are monotonically increasing function of supply voltage, varying V_{DD} allows throughput to be traded off for lower energy/operation, and vice-versa.

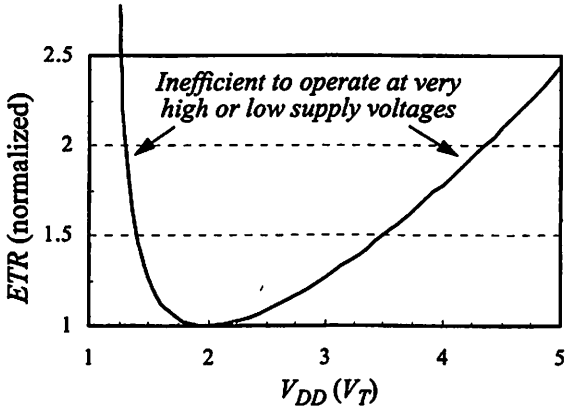


FIGURE 2.9 : *ETR* as a function of V_{DD}.

To compare designs over a larger operating range for the maximum throughput mode, a better metric is a plot of the energy/operation versus throughput. To make this plot, the supply voltage is varied from the minimum operating voltage (near V_T in many digital CMOS designs) to the maximum voltage (2.5-5V, depending on the technology), while energy/operation and throughput are measured. The energy/operation can then be plotted as a function of throughput, and the architecture is completely characterized over all possible throughput values.

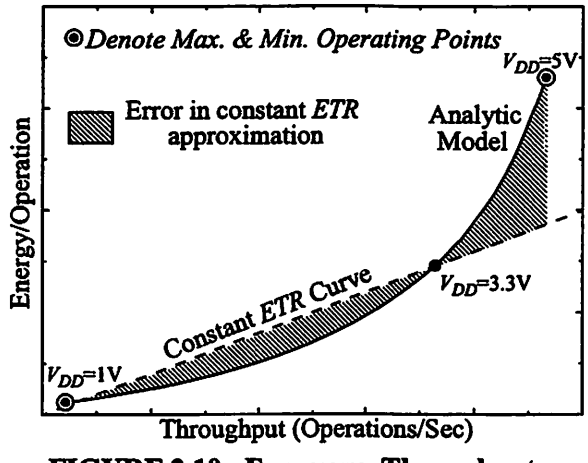


FIGURE 2.10 : Energy vs. Throughput

Using the *ETR* metric is equivalent to making a linear approximation to the actual energy/operation versus throughput curve. Figure 2.10 demonstrates the error incurred in using a constant *ETR* metric, which is calculated at a nominal supply voltage of 3.3V for this example. For architectures with similar throughput, a single

ETR value is a reasonable metric for energy efficiency; however, for designs optimized for vastly different values of throughput, a plot may be more useful, as Section 2.4.1 will demonstrate.

Using the throughput and energy models from Section 2.2, the *ETR* is:

$$ETR = \frac{C_L \cdot N_{gates} \cdot C_{EFF} \cdot V_{DD}^4}{k_v \cdot W \cdot (V_{DD} - V_T)^2} \quad \text{(EQ 2.20)}$$

However, this equation is not entirely intuitive in aiding in energy-efficient design, since the variables have several interdependencies. If the device width, W, is increased to reduce *ETR*, C_L and C_{EFF} will also increase, effectively increasing *ETR* when the gate capacitance begins to dominate the load capacitance. Similarly, if N_{gates} is reduced, this may come at the cost of increased C_L and/or C_{EFF}. Hence, individual parameters cannot be optimized in isolation, and their inter-dependencies must be taken into account by fully evaluating the *ETR* when optimizing a circuit for energy efficiency.

2.3.3 Burst Throughput Mode

Most single-user systems (e.g., stand-alone desktop computers, notebook computers, PDAs, etc.) spend only a fraction of the time performing useful computation. The rest of the time is spent idling between processes. However, when bursts of computation are demanded, the faster the throughput (or equivalently, response time), the better. This characterizes the burst throughput mode of computation in which most portable devices operate. The metric of energy efficiency used for this mode must balance the desire to minimize energy consumption, while both idling and computing, and to maximize peak throughput when computing.

Ideally, the processor's clock should track the periods of computation in this mode so that when an idle period is entered, the clock is immediately shut off. Then a good metric of energy efficiency is just *ETR*, as the energy consumed while idling has

been eliminated. However, this is not realistic in practice. Many processors do not having an energy saving mode and those that do so generally support only simple clock reduction/deactivation modes.

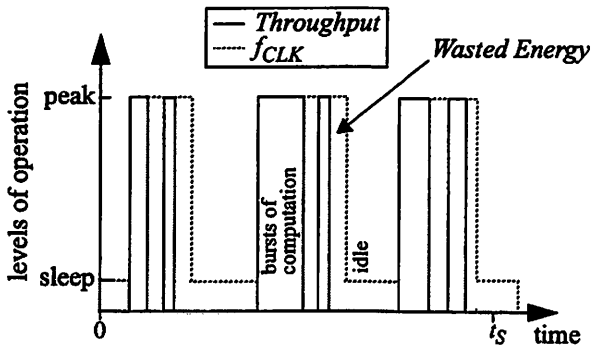


FIGURE 2.11 : Wasted energy due to idle

The hypothetical example depicted in Figure 2.11 contains a clock reduction (sleep) mode in which major sections of the processor are shut down. The shaded area indicates the processor's idle cycles in which energy is needlessly consumed, and whose magnitude is dependent upon whether the processor is operating in the "low-power" mode. The energy/operation while actively computing, E_{MAX}, and the amortized energy/operation while idling, E_{IDLE}, is:

$$E_{MAX} = \frac{Total\ Energy\ Consumed\ Computing}{Total\ Operations} \quad \text{(EQ 2.21)}$$

$$E_{IDLE} = \frac{Total\ Energy\ Consumed\ Idling}{Total\ Operations} \quad \text{(EQ 2.22)}$$

Total energy and total operations can be calculated over a large sample time period, t_S. T_{MAX} is the peak throughput during the bursts of computation (similar to that defined in Section 2.3.2), and T_{AVE} is the time-averaged throughput (total operations / t_S). If the time period t_S is sufficiently long that the operation characterizes the "average" computing demands of the user and/or target system environment yielding the average throughput (T_{AVE}), then a good metric of energy efficiency for the burst throughput mode is:

$$Metric|_{BURST} = BETR = \frac{E_{MAX} + E_{IDLE}}{T_{MAX}} \quad \text{(EQ 2.23)}$$

This metric will be called the Burst-mode *ETR* (*BETR*); it is similar to *ETR*, but also accounts for energy consumed while idling. A lower *BETR* represents a more energy-efficient solution.

Multiplying Equation 2.21 by the actual time computing [$t_S \cdot$ (fraction of time computing)], shows that E_{MAX} is the ratio of compute power dissipation to peak throughput T_{MAX}, as previously defined in Section 2.3.2. Thus, E_{MAX} is only a function of the hardware and can be measured by operating the processor at full utilization.

E_{IDLE}, however, is a function of t_S and T_{AVE}. The power consumed idling must be measured while the processor is operating under typical conditions, and T_{AVE} must be known to then calculate E_{IDLE}. However, expressing E_{IDLE} as a function of E_{MAX} better illustrates the conditions when idle energy consumption is significant. In doing so, E_{IDLE} will also be expressed as a function of the idle power dissipation, which is readily calculated and measured, as well as independent of t_S and T_{AVE}.

Equation 2.22 can be rewritten as:

$$E_{IDLE} = \frac{[Idle\ Power\ Dissipation][Time\ Idling]}{[Average\ Throughput][Sample\ Time]} \qquad \text{(EQ 2.24)}$$

With the Power-Down Efficiency, β, defined as:

$$\beta = \frac{Power\ dissipation\ while\ idling}{Power\ dissipation\ while\ computing} = \frac{P_{IDLE}}{P_{MAX}} \qquad \text{(EQ 2.25)}$$

E_{IDLE} can now be expressed as a function of E_{MAX}:

$$E_{IDLE} = \frac{[\beta \cdot E_{MAX} T_{MAX}] \cdot [(1 - T_{AVE}/T_{MAX}) t_S]}{[T_{AVE}] \cdot [t_S]} \qquad \text{(EQ 2.26)}$$

Equation 2.27 shows that idle energy consumption dominates total energy consumption when the fractional time spent computing (T_{AVE} / T_{MAX}) is less than the fractional power dissipation while idling (β).

$$BETR = ETR\left[1 + \beta\left(\frac{T_{MAX}}{T_{AVE}} - 1\right)\right], \quad T_{MAX} \geq T_{AVE} \qquad \text{(EQ 2.27)}$$

The *BETR* is a good metric of energy efficiency for all values of T_{AVE}, T_{MAX}, and β as illustrated below by analyzing the two limits of the *BETR* metric.

Idle Energy Consumption is Negligible ($\beta << T_{AVE}/T_{MAX}$): The metric should simplify to that found in the maximum throughput mode, since it is only during the bursts of computation that energy is consumed and operations performed. For negligible power dissipation during idle, the *BETR* metric in Equation 2.27 degenerates to the *ETR*, as expected. For perfect power-down ($\beta = 0$) or high user-demanded throughput ($T_{MAX} = T_{AVE}$), the *BETR* is exactly the *ETR*.

Idle Energy Consumption Dominates ($\beta >> T_{AVE}/T_{MAX}$): The energy efficiency should increase by either reducing the idle energy/operation while maintaining constant throughput, or by increasing the throughput while keeping idle energy/operation constant. While it might be expected that these are independent optimizations, E_{IDLE} may be related back to E_{MAX} and the throughput by β since T_{AVE} is fixed:

$$\frac{E_{IDLE}}{E_{MAX}} \cong \frac{P_{IDLE}/T_{AVE}}{P_{MAX}/T_{MAX}} = \beta \cdot \frac{T_{MAX}}{T_{AVE}} \qquad \text{(EQ 2.28)}$$

Expressing E_{IDLE} as a function of E_{MAX} yields:

$$BETR \cong \frac{\beta \cdot E_{MAX}}{T_{AVE}} \qquad \textit{(Idle Energy Dominates)} \qquad \text{(EQ 2.29)}$$

If β remains constant for varying throughput (and E_{MAX} stays constant), then E_{IDLE} scales with throughput as shown in Equation 2.28. Thus, the *BETR* becomes an energy/operation minimization similar to the fixed throughput mode. However, β may vary with throughput, as will be analyzed further in Section 5.2.5.

2.3.4 Energy Efficiency for Practical Designs

As mentioned earlier, the *BETR* metric measures the energy efficiency of processor systems. Unfortunately, information on the system's average throughput (T_{AVE}) is required to utilize this metric, which is application specific. Thus, the *BETR*

metric cannot be used to describe the energy efficiency of a processor in general terms, but requires the specification of a target application, or class of related applications. An example application is the InfoPad, as described in Section 2.1.3, in which the processor system is responsible for packet-level network control on the pad and has an average throughput requirement of 0.8 MIPS. If the video decompression was implemented by the processor rather than the custom chip-set, then the average throughput would increase to approximately 11 MIPS.

So that energy-efficient design techniques can be discussed independent of the final application, the *BETR* metric's subcomponents, *ETR* and E_{IDLE}, will be discussed individually.

2.4 Energy Efficient Design Principles

Four examples are presented in this section to demonstrate how energy efficiency can be properly quantified. In the process, four design principles follow from the optimization of the previously defined metrics: a high-performance processor is generally energy-efficient; idle energy consumption limits the energy efficiency for high-throughput operation; reducing the clock frequency is never energy efficient; and dynamic voltage scaling is very energy efficient.

2.4.1 High Performance is Energy Efficient

Table 2.1 lists two processors that are available today – the ARM710 targets the low-power market, and the R4700 targets the mid-range workstation market, and both are fabricated in similar 0.6μm technologies, facilitating an equal comparison. The measure of throughput used is SPECint92. A commonly-used metric for measuring energy efficiency is SPECint92/Watt (or SPECint95/Watt, Dhrystones/Watt, MIPS/Watt, etc.). The ARM710 processor has a SPECint92/Watt five times greater than the R4700's, and the claim then follows that it is "five times as energy efficient".

However, this metric only compares operations/energy, and does not weight the fact that the ARM710 has only 15% of the performance as measured by SPECint92.

TABLE 2.1 Comparison of two processors [arm94][idt95].

Processor	SPECint92 (T_{MAX})	Power (Watts)	Supply voltage, V_{DD} (Volts)	SPECint92/Watt ($1/E_{MAX}$)	*ETR* (10^{-3})
R4700	130	4.0	3.3	33	0.24
ARM710	20	0.12	3.3	167	0.30

The *ETR* (Watts/SPECint92^2) metric indicates that the R4700 is actually more energy efficient than the ARM710. To quantify the efficiency increase, the plot of energy/operation versus throughput in Figure 2.12 is used because it better tracks the R4700's energy at the low throughput values. The plot was generated from the throughput and energy/operation models in Section 2.2.

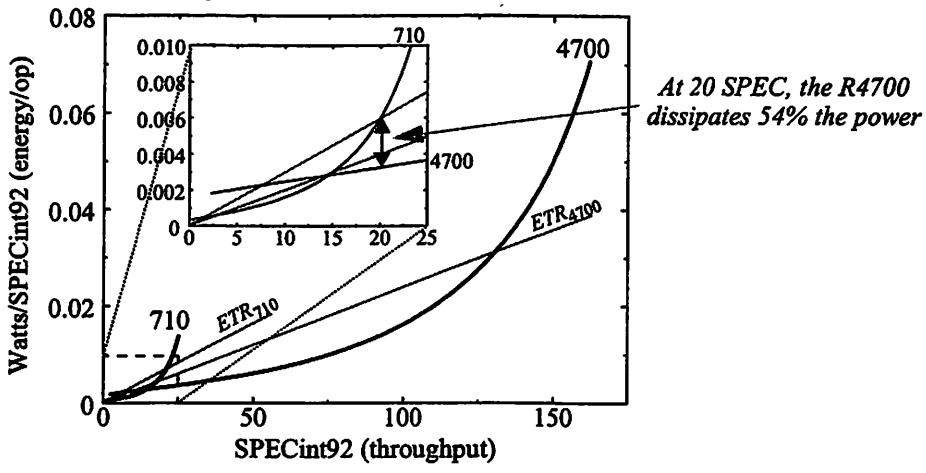


FIGURE 2.12 : Energy vs. Throughput of R4700 and ARM710.

According to the plot, the R4700 would dissipate 65mW at 20 SPECint92, or about 1/2 of the ARM710's power, despite the low V_{DD} ($1.5 \cdot V_T$) for the R4700. Conversely, the R4700 can deliver 30 SPECint92 at 120mW ($V_{DD} = 1.7 \cdot V_T$), or 150% of the ARM710's throughput.

This does assume that the R4700 processor has been designed so that it can operate at these low supply voltages. If the lower bound on operating voltage is greater than $1.7 \cdot V_T$, then the ARM710 would be more energy efficient in delivering the

20 SPECint92 than the R4700. Typically, a processor is rated for a fixed standard supply voltage (3.3V or 5.0V) with a ±10% tolerance. However, many processors can operate over a much larger range of supply voltages (e.g., 2.7-5.5V for the ARM710 [arm94], 2.0-3.3V for the Intel486GX [inte95]). The processor can operate at a non-standard supply voltage by using a high-efficiency, low-voltage DC-DC converter to generate the appropriate supply voltage [stra94].

While the *ETR* correctly predicted the more energy-efficient processor at 20 SPECint92, it is important to note that the R4700 is not more energy efficient for all values of SPECint92, as the *ETR* metric would indicate. Because the nominal throughput of the processors is vastly different, the Energy/Operation versus Throughput metric better tracks the efficiency, and indicates a cross-over throughput of 14.5 SPECint92. Below this value, the ARM710 becomes more energy efficient.

2.4.2 Fast Operation Can Limit Energy Efficiency

If the user demands a fast response time, rather than reducing the voltage, as was done in Section 2.4.1, the processor can be left at the nominal supply voltage, and shut down when it is not needed.

For example, assume the target application has a T_{AVE} of 20 SPECint92, and both the ARM710 and R4700 have a β factor of 0.2. If the processors' V_{DD} is left at 3.3V, The ARM710's *BETR* is exactly equal to its *ETR* value, which is 3.0×10^{-4}. It remains the same because it never idles. The R4700, on the other hand, spends 85% $(1 - T_{AVE}/T_{MAX})$ of the time idling, and its *BETR* is 5.0×10^{-4}. Thus, for this scenario, the ARM710 is nearly twice as energy efficient.

However, if the R4700's β can be reduced down to 0.02, then the *BETR* of the R4700 becomes 2.66×10^{-4}, and it is once again the more energy-efficient solution. For this example, the cross-over value of β is 0.045.

This example demonstrates how important it is to use the *BETR* metric instead of the *ETR* metric if the target application's idle time is significant (i.e., T_{AVE} can be characterized and is significantly below T_{MAX}). For the above example, a β for the R4700 greater than 0.045 leads the metrics to disagree on which is the more energy-efficient solution. One might argue that the supply voltage can always be reduced on the R4700 so that it is more energy efficient for any required throughput. This is true if the dynamic range of the R4700 is as indicated in Figure 2.12. However, if some internal logic limited the value that V_{DD} could be dropped, then the lower bound on the R4700's throughput would be located at a much higher value. Thus, finite β can degrade the energy efficiency of a high-throughput processor, due to excessive idle power dissipation.

2.4.3 Clock Frequency Reduction is Never Energy Efficient

A common fallacy is that reducing the clock frequency, f_{CLK}, is energy efficient. Reducing f_{CLK} does reduce power dissipation, but it does not increase energy efficiency. When compute energy consumption dominates idle energy consumption, it actually increases energy efficiency. At best, when idle energy consumption is dominant, it allows an energy-throughput trade-off. The relative amount of time spent idling versus computing is an important consideration in determining the effect of clock frequency reduction on energy efficiency.

Compute energy consumption dominates ($E_{MAX} >> E_{IDLE}$): Since compute energy consumption is independent of f_{CLK}, and throughput scales proportionally with f_{CLK}, decreasing the clock frequency increases the *ETR*, and thereby reduces energy efficiency. Halving f_{CLK} is equivalent to doubling the computation time, while maintaining constant computation per battery life, which is clearly energy inefficient.

Idle energy consumption dominates ($E_{IDLE} >> E_{MAX}$): Clock reduction may trade-off throughput and energy/operation, but only when the power-down efficiency, β,

is independent of throughput such that E_{IDLE} scales with throughput. When this is so, halving f_{CLK} will double the computation time, but will also double the amount of computation per battery life, since E_{IDLE} has been halved. If the currently executing process can tolerate throughput degradation, then this may be a reasonable trade-off. If β is inversely proportional to throughput, however, then reducing f_{CLK} does not affect the total energy consumption, and the energy efficiency drops.

As shown in Table 2.2, reducing the clock frequency reduces energy efficiency in two of the three possible operating conditions. In the third operating condition, the efficiency merely remains unchanged. Thus, clock frequency reduction is never energy efficient.

TABLE 2.2 Impact of Clock Frequency Reduction on Energy Efficiency.

Operating Conditions:	Compute Energy Consumption Dominates	Idle Energy Consumption Dominates	
		β independent of throughput	β inversely proportional to throughput
Throughput	decreases	decreases	decreases
Energy	unchanged	decreases	unchanged
Energy Efficiency (1 / *BETR*)	***decreases***	***unchanged***	***decreases***

2.4.4 Dynamic Voltage Scaling is Energy Efficient

If V_{DD} were to track f_{CLK}, however, so that the critical path delay remains inversely equal to the clock frequency, then constant energy efficiency could be maintained as f_{CLK} is varied. This is equivalent to V_{DD} scaling (Section 2.3.2) except that it is done dynamically during processor operation. If E_{IDLE} is present and dominates the total energy consumption, then simultaneous f_{CLK} and V_{DD} reduction during periods of idle will yield a more energy-efficient solution.

Even when idle energy consumption is negligible, dynamic voltage scaling provides significant wins. Figure 2.13 plots a sample usage pattern of desired

throughput, with the delivered throughput super-imposed on top. For background and high-latency tasks, the supply voltage can be reduced so that just enough throughput is delivered, which minimizes energy consumption.

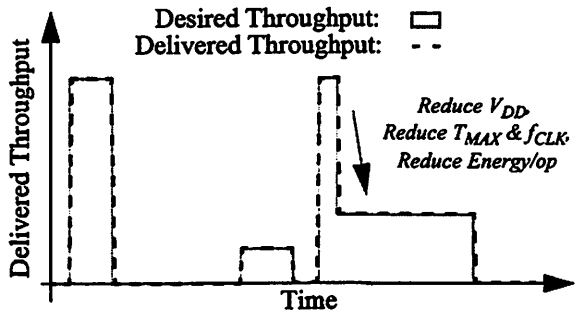


FIGURE 2.13 : Dynamic Voltage Scaling.

For applications that require maximum deliverable throughput only a small fraction of the time, dynamic voltage scaling provides a significant energy efficiency improvement. For the R4700 processor, the peak throughput is 130 SPECint92. Given a target application where the desired throughput is either a fast 130 SPECint92 or a slow 13 SPECint92, Table 2.3 lists the peak throughput, average energy/operation, and effective *ETR* depending on the fraction of time spent in the fast mode. For each category of throughput the total number of operations completed are the same so that the relative changes in battery life can be evenly compared. When that fraction becomes small, the processor's peak throughput is still set by the fast mode, while the average energy consumed per operation is set by the slower mode. Thus, the best of both extremes can be achieved. For simplicity, this examples assumes that idle energy consumption is always negligible.

TABLE 2.3 Benefits of Dynamic Voltage Scaling.

Throughput:	Time spent operating in:			T_{MAX} (SPECint92)	E_{MAX} (W/SPECint92)	*ETR* (10^{-6})	Normalized Battery Life
	Fast Mode	Slow mode	Idle Mode				
Always full-speed	10%	0%	90%	130	0.031	237	1 hr.
Sometimes full-speed	1%	90%	9%	130	0.006	45.0	5.3 hrs.
Rarely full-speed	0.1%	99%	0.9%	130	0.003	25.8	9.2 hrs.

As shown in Table 2.3, the battery run-time can be improved by up to a factor of 10x. In most portable devices (e.g. notebook computers, PDAs, etc.), peak throughput is typically used only a small fraction of the time, such that this energy-efficiency improvement is readily achievable. Although dynamically varying V_{DD} and f_{CLK} in a processor system may seem extraordinarily difficult to accomplish, Chapter 3 will demonstrate that dynamic voltage scaling is a relatively straightforward and simple technique to implement.

3 Dynamic Voltage Scaling

Dynamic Voltage Scaling (DVS) can significantly improve processor energy-efficiency for burst-mode operation. This technique can decrease the system's average energy consumption while computing, E_{MAX}, by more than 10x, without sacrificing perceived throughput, T_{MAX}, by exploiting the time-varying computational load that is commonly found in portable electronic devices. By dynamically varying both the processor's clock frequency and supply voltage in response to computational load demands, the processor always operates at just the desired performance level while consuming the minimal amount of energy. Since T_{MAX} remains constant while E_{MAX} decreases, both the *ETR* and *BETR* metrics will scale down by a proportional amount, providing a potential increase in energy efficiency in excess of 10x.

There are three key components for implementing DVS in a general-purpose microprocessor system: an operating system that can intelligently vary the processor speed, a regulation loop that can generate the minimum voltage required for the desired speed, and a microprocessor that can operate over a wide voltage range.

This chapter focuses on the implementation methodology for DVS, the new system constraints imposed by DVS, and the impact of DVS on the hardware design methodology. This methodology was validated by a prototype embedded processor system that successfully implements DVS, and is described further in Chapter 7.

3.1 Overview

Digital CMOS circuits are very amenable to implementing DVS, as their performance and energy consumption scale together over a wide range of supply voltage. Although the maximum supply voltage drops with improved process technology, thereby reducing this range, so does the device threshold voltage, such that DVS will continue to be a viable technique for future process technologies. Furthermore, DVS provides a solution to the leakage problem of low threshold-voltage processes by scaling leakage current with supply voltage.

3.1.1 Voltage Scaling Effects on Circuit Delay

CMOS circuit delay tracks very well over supply voltage, as shown in Figure 3.1. Four example circuits are shown, ranging in complexity from a simple inverter to a complex SRAM design that consists of an address decoder, memory cell array, sense amplifier, output buffer, and control sequencing logic. The maximum clock speed is just the inverse of the critical path delay, which was calculated via a SPICE simulation and then normalized at 4V (the SPICE data for the SRAM is from [burs97]).

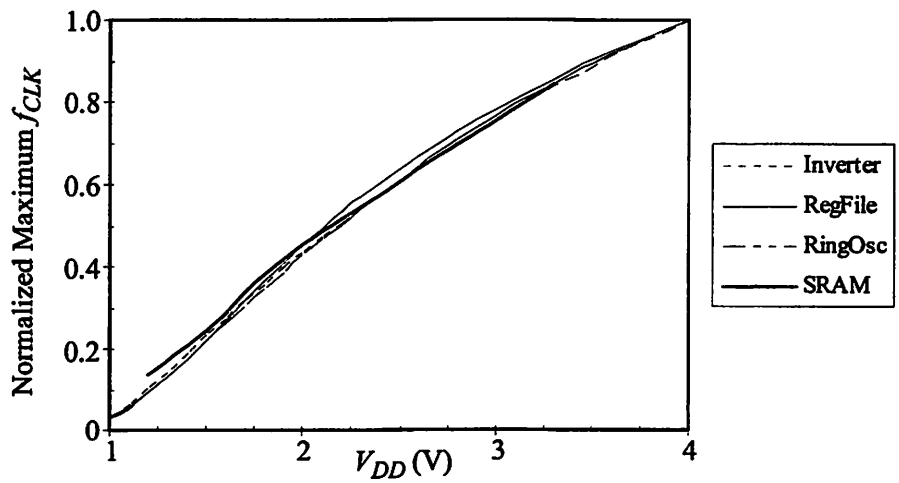


FIGURE 3.1 : Various Circuit Delays vs. Supply Voltage

The inverter, ring oscillator, and register file all vary less than 10% over the full range of supply voltage, V_{DD}. These three circuits are a good cross-representation

of the bulk of CMOS circuits, both in logic style and complexity. The SRAM circuit, which differs from the others because NMOS devices dominate its critical path delay, runs faster at lower voltage because for our 0.6μm process, $V_{Tn} < V_{Tp}$. However, the speed variation at 1.2V is still only 25%, which is insignificant compared to the 10x overall reduction in maximum speed from 4V. More importantly, the deviation is in the positive direction; because the SRAM circuit runs faster at lower V_{DD}, it will not be a limiting factor of the chip's speed at low V_{DD}.

By using a ring oscillator to generate the clock signal, the clock frequency can be scaled lock-step with V_{DD}, enabling proper operation of the processor over the full range of V_{DD} through a closed-loop control system.

3.1.2 Maximum Energy Efficiency Improvement.

Figure 3.2 demonstrates the possible energy-efficiency improvement of DVS. Starting at the nominal V_{DD} operating point of 3.3V, when the clock frequency, f_{CLK}, is reduced, there is a proportional decrease in throughput. When this is done at constant V_{DD}, there is no reduction in energy/operation. However, if V_{DD} is scaled lock-step with f_{CLK}, then the lower curve is traversed, yielding more than a 10x energy reduction at low voltage.

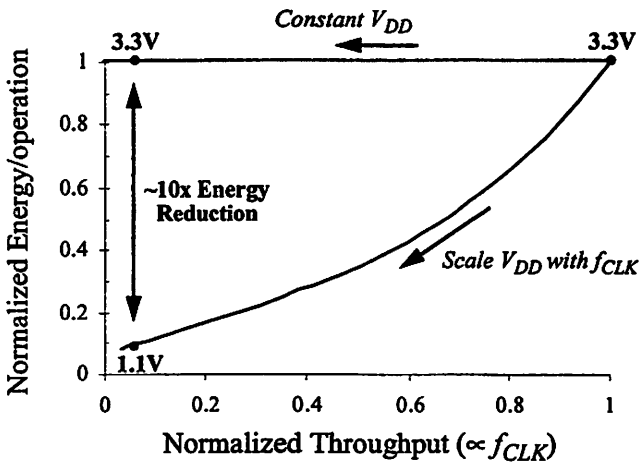


FIGURE 3.2 : Scaling V_{DD} with f_{CLK}.

The ability to dynamically traverse this curve is how DVS radically improves energy efficiency. For the processor described in Chapter 7, the lower operating point is 6 MIPS @ 0.27 mW/MIPS (ETR = 45 μW/MIPS^2), and the upper operating point is 85 MIPS @ 2.8 mW/MIPS (ETR = 33 μW/MIPS^2). However, if peak throughput is only occasionally demanded, then the processor can deliver a peak throughput of 85 MIPS, while the average energy/operation can be as low as 0.27 mW/MIPS. This yields an ETR of 3.2 μW/MIPS^2, which is more than a 10x improvement in energy efficiency.

Figure 3.3 plots the normalized battery run-time, which is inversely proportional to energy/operation, as a function of the fractional amount of computation performed at low throughput for the above processor. While a moderate run-time increase (22%) can be achieved with only 20% of the computation at low throughput, DVS yields significant increases when more of the computation can be run at low throughput, with the upper limit in excess of a 10x increase in battery run-time, or equivalently, more than a 10x reduction in energy/operation.

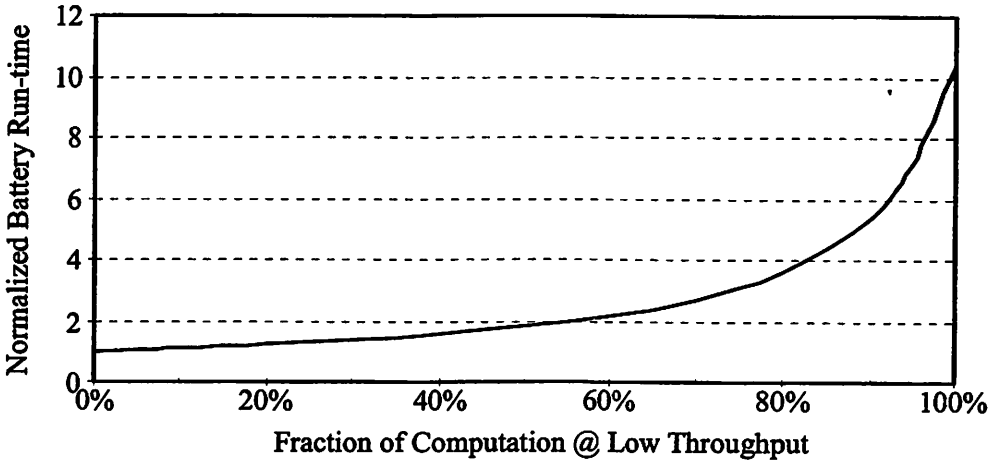


FIGURE 3.3 : Battery Run-time vs. Workload

3.1.3 Essential Components

A typical processor system is powered by a voltage regulator which outputs a fixed voltage. However, the implementation of DVS requires a voltage converter that can dynamically adjust its output voltage when requested by the processor to do so.

With no commercial dynamic voltage converters available, a prototype converter was designed and implemented [stra98]. This functionality can also be implemented with discrete commercial components, but with much lower conversion efficiency than that of the custom prototype.

Another essential component is a ring oscillator matched to the processor's critical paths, such that as the critical paths vary over V_{DD}, so will the processor clock frequency. This is best achieved by having a ring oscillator on the processor, which will then track the critical paths over process and temperature.

The processor itself must be designed to operate over the full range of voltage supply, which places restrictions on the types of circuits that can be used and impacts processor verification, as described in Section 3.3. Additionally, the processor must be able to properly operate while V_{DD} is varying, as detailed in Section 3.4.

The last essential component is a DVS-aware operating system. The hardware itself has no knowledge of the priority of the currently executing task, since this information only resides within the operating system scheduler. Hence, to deliver the significant increase in energy efficiency afforded by DVS, the operating system must be able to intelligently vary V_{DD} and f_{CLK} as a function of desired throughput, which is further described in Section 3.5.

3.1.4 Fundamental Trade-Off

Processors generally operate at a fixed voltage, and require a regulator to tightly control voltage supply variation. The processor produces large current spikes for which the regulator's output capacitor supplies the charge. Hence, a large output capacitor on the regulator is desirable to minimize ripple on V_{DD}. A large capacitor also helps to maximize the regulator's conversion efficiency by reducing the voltage variation at the output of the regulator.

However, the voltage converter required for DVS is fundamentally different from a standard voltage regulator because in addition to regulating voltage for a given clock frequency, it must also change the operating voltage when a new clock frequency is requested. To minimize the speed and energy consumption of this voltage transition, a small output capacitor on the converter is desirable, in contrast to the supply ripple requirements.

Thus, the fundamental trade-off in a DVS system is between good voltage regulation and fast/efficient dynamic voltage conversion. As will be shown in Section 3.2, it is possible to optimize the size of this capacitor to balance the requirements for good voltage regulation with the requirements for a good dynamic voltage conversion.

3.1.5 Scalability with Technology

For DVS to provide significant energy efficiency improvement, the process technology must be able to operate over a wide range of voltage, such that the throughput and energy consumption can appreciably vary.

The lower bound on voltage is set by the larger of V_{Tn} and V_{Tp}, beyond which the MOSFETs begin operating in the subthreshold region, and their delay increases exponentially [pier96]. A more practical limit is ~100mV above $\max(V_{Tp},V_{Tn})$, to provide an operating margin for preventing the MOSFETs from entering this region.

The upper bound on voltage is determined by gate-oxide breakdown [mull86]. For our 0.6μm process, this is only 6.3V. To provide a margin of safety, a process has a rated maximum voltage of around one-half of the gate-oxide breakdown voltage; for the 0.6μm process, the rated maximum voltage is 3.3V. While the MOSFETs can be operated at a higher voltage, it is generally not recommended for long-term gate-oxide reliability.

As process technology advances, the reduction in gate-oxide thickness necessitates a reduction in the rated maximum supply voltage. However, to maintain MOSFET performance, their threshold voltages have also been reduced, as shown by the sampling of 24 process technologies in Figure 3.4. Scaling V_T with V_{DD} maintains a large range of energy consumption (V_{DD}^2 / (V_T + 100mV)^2), anywhere from 10x to over 30x. A future 0.10μm process may only have a rated voltage of 1.2V, but with a V_T of 0.3V and 50mV of operating margin, the possible energy range is still 11.8x. Since throughput scales by a similar order of magnitude as does energy consumption, DVS is still quite applicable to even deep-submicron process technologies.

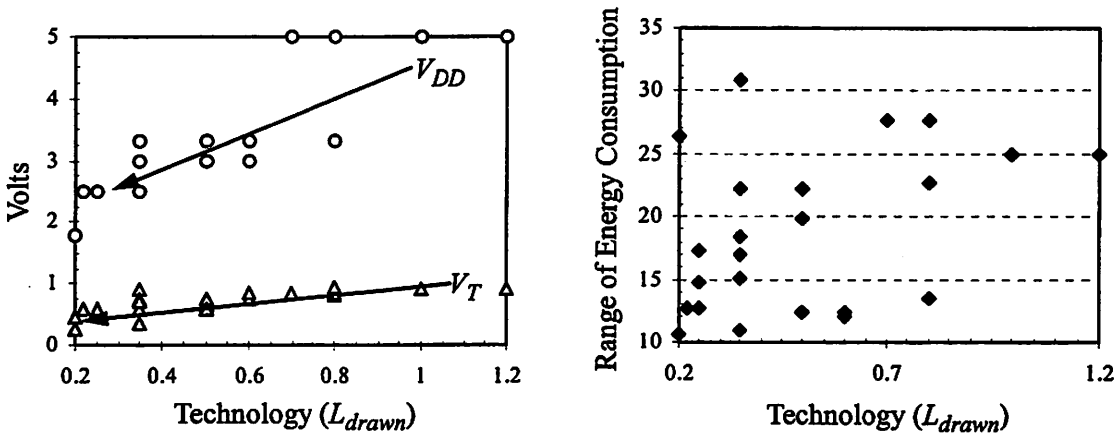


FIGURE 3.4 : V_{DD}, V_T, and Range of Energy Consumption vs. Process Technology.

3.2 Converter Feedback Loop

The voltage converter loop is a non-linear negative-feedback loop. The steady-state operation forces the processor clock, f_{CLK}, to be:

$$f_{CLK} = F_{DES} \cdot (1\,MHz) \qquad \text{(EQ 3.1)}$$

where F_{DES} is the desired frequency in MHz, and is stored as a digital word by the processor hardware. Thus, the processor requires no knowledge of the actual supply voltage. It simply adjusts f_{CLK} and V_{DD} by requesting a new operating frequency.

3.2.1 Buck Converter

The loop is built around a buck-converter (Figure 3.5) which is very amenable to high-efficiency, low-voltage regulation [stra94]. Using a digital pulse-width modulation (PWM) algorithm, the buck-converter converts the battery voltage, V_{BAT}, to the desired output voltage, V_{DD}, as a function of the pulse duty cycle, D:

$$V_{DD} = V_{BAT} \cdot D \qquad \text{(EQ 3.2)}$$

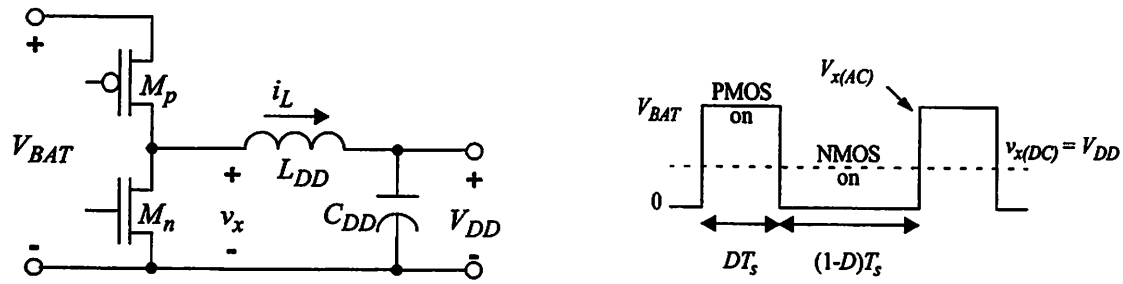


FIGURE 3.5 : Buck Converter Design and Operation.

During the positive pulse, the PMOS power FET, M_p, is turned on and the inductor current, i_L, begins ramping up, pushing charge onto the capacitor, C_{DD}. During the negative pulse, the rectifying NMOS power FET, M_n, is turned on and i_L begins ramping down. The LC tank filters V_x so that a steady-state DC voltage appears across the capacitor as V_{DD}; the inductor, L_{DD}, absorbs the voltage differential during the switching transient. The power FETs are only switched on when $V_x = V_{BAT}$ (PMOS) or $V_x = 0$ (NMOS) to maintain a minimal drain-source voltage drop in order to minimize energy loss in the power FETs [stra98]. T_S is the fixed clock period of the converter.

To improve the converter's efficiency at low voltage and/or light load, the converter loop also implements a pulse-frequency modulation (PFM) algorithm [stra98]. At low voltage and/or light load, the processor's energy consumption is very small, and so too is the current it draws from V_{DD}. Rather than enable the PMOS for an infinitesimally small amount of time to replace the small amount of charge removed

from C_{DD} by the processor, the converter is selectively enabled through pulse-skipping, by which, in a given period T_S, if the voltage drop on V_{DD} is sufficiently small, the converter is simply disabled. The conversion efficiency is greatly increased due to the saved energy cost of enabling the power FETs, but comes with the penalty of increased voltage ripple.

3.2.2 Loop Architecture

The full converter loop architecture is shown in Figure 3.6. The output of the ring oscillator, f_{CLK}, clocks a counter which is reset at 1 MHz intervals. This provides the quantized digital word, F_{MEAS}, which is the measured clock frequency in MHz. This value is subtracted from the desired clock frequency, F_{DES}, to generate an error frequency value, F_{ERR}. A positive value indicates a higher voltage is required to increase f_{CLK}, and a negative value indicates that the voltage is too high.

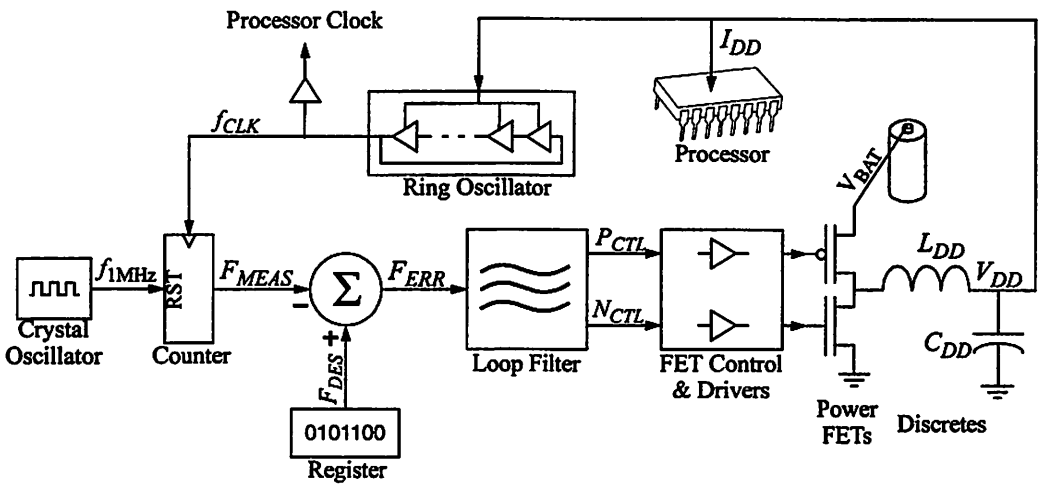


FIGURE 3.6 : DVS Voltage Converter Loop Architecture.

The loop filter does two important functions. First, it converts the frequency error into an equivalent voltage error via a hardware look-up table. Next, it converts the equivalent voltage error into an update command for the power FETS through the hybrid PWM-PFM scheme described in the previous section. When $-3 \leq F_{ERR} \leq 0$ indicating that the voltage is slightly high, the pulse-skipping algorithm disables the

converter for the current clock cycle, allowing the processor to discharge V_{DD}. Any other value of F_{ERR} enables the converter for the current cycle through the control signals P_{CTL} and N_{CTL} [stra98].

These two control signals are converted to power FET enable signals, which are buffered to drive the large gate capacitance of the power FETs. The buck converter produces an output voltage V_{DD} which is sent back to the ring oscillator, closing the loop. In addition, the processor is powered by V_{DD}, so it draws a time-varying current I_{DD} from the output capacitor.

3.2.3 Loop Stability

The external filter components, shown in Figure 3.7, primarily dictate the frequency response of the converter loop. R_{DD} is the effective resistance of the V_{DD} load, and varies as a function of V_{DD}.

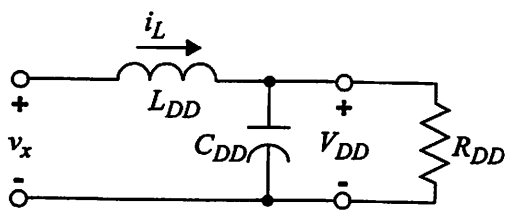


FIGURE 3.7 : Converter Loop RLC Filter.

In a typical buck converter, this filter has two poles, due to the capacitor voltage, V_{DD}, and inductor current, i_L, state variables. However, in this system, charge is delivered to the capacitor, C_{DD}, in discrete quantities, thereby ensuring that i_L starts and ends each cycle at zero which eliminates it as a state variable. Although operating in this discontinuous mode increases the voltage ripple on C_{DD}, it reduces this filter to a one pole system, whose pole is set by R_{DD} and C_{DD}, as shown in Figure 3.8.

There is a sampling delay introduced by the front-end clock quantizer, which places another pole around 1 MHz. As V_{DD} increases, R_{DD} decreases, and the dominant pole moves higher in frequency, potentially resulting in instability. For the system

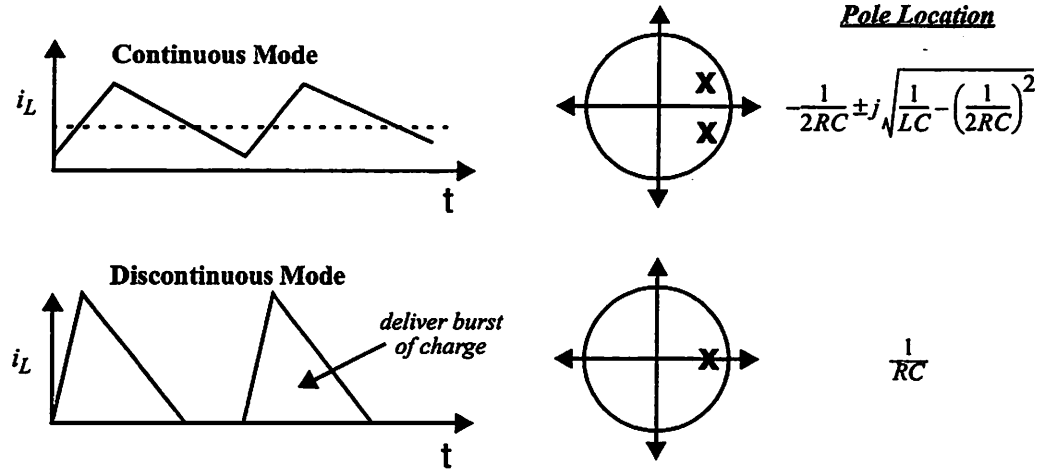


FIGURE 3.8 : Reducing the Buck Converter to a One Pole System

implementation in Chapter 7, peak current was 125mA at 4V so that the dominant pole is a maximum of 7kHz and the loop gain is less than one at 1 MHz, thereby ensuring system stability.

3.2.4 Software Interface

Changing the operating point of the converter loop is done by specifying a new frequency, abstracting away the actual voltage required to meet that frequency. This is quite suitable to the operating system for which voltage is meaningless. Desired frequency can simply be calculated as:

$$F_{DES} = \frac{Estimated\ Workload}{Time\ to\ Completion} \qquad \text{(EQ 3.3)}$$

where the estimated workload is measured in processor clock cycles, and the time to completion is derived from the time constraints of the active software process.

Since F_{DES} is specified as a digital word, it is implemented as a writable register, which must be placed in the visible instruction set architecture (ISA) to make it accessible to software. For the prototype system (Chapter 7), this register (CP14R4) was placed along with other programmable system state into the System Coprocessor. When the software writes to CP14R4, V_{DD} and f_{CLK} will immediately begin to adjust to

the desired levels, providing the software with direct and full control over the voltage converter system.

Since the processor in single-user systems is quite often idling, waiting for further user input, reducing idle energy consumption is important to improve the overall processor system energy efficiency. Since the operating system is aware of when the processor is idling, it will issue a processor halt instruction before these idle periods. To improve the idle energy efficiency, the operating system simply needs to set the desired clock frequency to a minimum before issuing the halt instruction. Then, the processor will be operating at minimum voltage, and minimum energy consumption. When user input is detected by the operating system, it can restart the processor, and restore the desired frequency to whatever the value was before halting.

If user-input is detected via a processor interrupt, it may be many cycles before the operating system can restore the desired frequency. To reduce this latency, the prototype processor changes the desired frequency when an interrupt occurs. By implementing the frequency change in hardware, which only requires an additional 8-bit register, the speed can be altered immediately.

3.2.5 Clock Generation

A significant benefit of the converter loop architecture is that it provides clock generation for the processor, which is simply a buffered version of f_{CLK}. The only external circuits required is a 1 MHz oscillator, which can be implemented with little power dissipation. The power dissipated by the oscillator itself can be <10μW [aebi97]. The power dissipated driving the clock signal on the printed-circuit board (PCB) is:

$$P_{PCB} = f_{CLK} \cdot C_L \cdot V_{DD}^2 \qquad \text{(EQ 3.4)}$$

For a C_L of 10pF, and a V_{DD} of 3.3V, the power dissipation for driving this 1 MHz clock signal is 100μW.

A typical processor either generates the f_{CLK} on chip via a phase-locked loop (PLL), or uses an externally generated signal. For a 100 MHz clock signal, the power driving the 10pF would be 11mW at 3.3V, and the oscillator itself can add another 10-100mW. Most mid-to-high performance processors have an on-chip PLL to generate the processor clock signal. But even the lowest reported power dissipation is 1.5mW, and still requires an external 3.68 MHz crystal oscillator [mont96].

In contrast, the ring-oscillator for the converter loop is the equivalent of 33 gates switching every cycle; in our 0.6μm process, this is approximately 1pF. The power dissipation of the ring oscillator scales with f_{CLK} and V_{DD}. At the low corner of 1.1V and 8 MHz, the power dissipated is only 10μW; this is 10x lower than conventional clock generation approaches, even taking the required external 1 MHz crystal oscillator into account. In addition, this capacitance will also scale down in technology so that in better process technology, the power dissipation will be lower for a given f_{CLK} and V_{DD}. Further reduction can be achieved by integrating the 1 MHz oscillator circuit on-chip, leaving only the crystal external to the chip, which would eliminate the power dissipation for driving the external 1 MHz clock signal.

3.2.6 Conversion Efficiency

The efficiency of a voltage regulator is defined as:

$$\eta = \frac{\textit{Power Delivered to Load}}{\textit{Total Power Dissipation}} \qquad \text{(EQ 3.5)}$$

with 100% being the maximum efficiency possible, in which no power is lost in delivering energy to the load circuits. The buck converter is very efficient at voltage conversion, with efficiencies typically in the 90-95% range [stra94]. While it can be designed methodically for a fixed operating voltage, the difficulty arises in designing for this efficiency across a range of voltage and current loads. Several techniques have been developed for the converter loop design to improve the efficiency over this broad range of operating conditions [stra98].

The loop filter PWM-PFM algorithm will not deliver charge when $-3 \leq F_{ERR} \leq 0$. For low voltage and/or light load conditions, when little charge is being drawn from V_{DD}, the loop filter stops activation of the power FETs which are the largest source of loss. Only one out of N cycles generates an "on" pulse, where N can be as high as 100 cycles.

The entire front-end is digital, which includes all the circuits starting from V_{DD} up to the generation of F_{ERR}. When $-3 \leq F_{ERR} \leq 0$, these are the only circuits actively operating and dissipating power. By taking their variable delay over voltage into account during the design of the loop, they can all be powered from V_{DD}, instead of V_{BAT}. Thus, the power of these circuits, which are continuously running, scales with the current V_{DD} operating point, so that at low voltage, their power dissipation becomes insignificant.

To improve efficiency while the buck converter is actively operating, the power FETs are comprised of multiple parallel FETs. Then, the actual FET size is dynamically varied to minimize loss over the range of operating conditions [stra98].

The combination of these techniques provides an efficiency of 80-95% while the processor is actively operating over the range of voltage and current load, and has negligible power loss while the processor is idling. Chapter 7 describes the energy efficiency of a prototype implementation in further detail.

3.2.7 New Performance Metrics

In addition to the supply ripple and conversion efficiency performance metrics of a standard voltage regulator, the DVS converter introduces two new performance metrics: transition time and transition energy. For a large voltage change ($V_{DD1} \rightarrow V_{DD2}$), the transition time is:

$$t_{TRAN} \approx \frac{2 \cdot C_{DD}}{I_{MAX}} \cdot |V_{DD2} - V_{DD1}| \qquad \text{(EQ 3.6)}$$

where I_{MAX} is the maximum output current of the converter, and the factor of 2 exists because the current pulses are triangular. In practice, t_{TRAN} will be slightly longer for a low-to-high voltage transition because the actual current charging C_{DD} is I_{MAX} - $I_{DD}(V_{DD})$. The energy consumed during this transition is:

$$E_{TRAN} = (1-\eta) \cdot C_{DD} \cdot \left| V_{DD1}^2 - V_{DD2}^2 \right| \qquad \text{(EQ 3.7)}$$

Since both transition time and transition energy are proportional to C_{DD}, minimizing C_{DD} yields a faster and more energy-efficient voltage converter.

To gauge how the transition energy impacts the overall system energy consumption, it is more intuitive to compare the power dissipation which factors in the frequency of voltage transitions and level of processor performance. Given a frequency, f_{VDD}, at which the system makes voltage transitions, the transition power dissipation is:

$$P_{TRAN} = E_{TRAN} \cdot f_{VDD} = (1-\eta) \cdot C_{DD} \cdot \left| V_{DD1}^2 - V_{DD2}^2 \right| \cdot f_{VDD} \qquad \text{(EQ 3.8)}$$

Figure 3.9 demonstrates how transition time (t_{TRAN}) and transition power dissipation (P_{TRAN}) vary with C_{DD} for the maximum 1.2-3.8V voltage transition of the prototype system, which has I_{MAX} = 1A, η = 90%. P_{TRAN} is shown for three different values of f_{VDD}. Also plotted is the minimum prototype system power dissipation not including P_{TRAN}, and sets the threshold below which P_{TRAN} should remain so that it does not dominate the total system power dissipation. A typical C_{DD} value for low-voltage/low-power voltage regulators is 100µF. This gives a t_{TRAN} in excess of 500µs which precludes any real-time control or fast interrupt response time, and only allows very coarse speed control. For this value of C_{DD}, an f_{VDD} on the order of a context switch (30-100Hz) will cause the transition power to dominate the system power (55-80% of the total power).

Thus, existing voltage regulators make very poor voltage converters due to their large C_{DD}, which needs to be reduced by at least 10x. Using the converter loop,

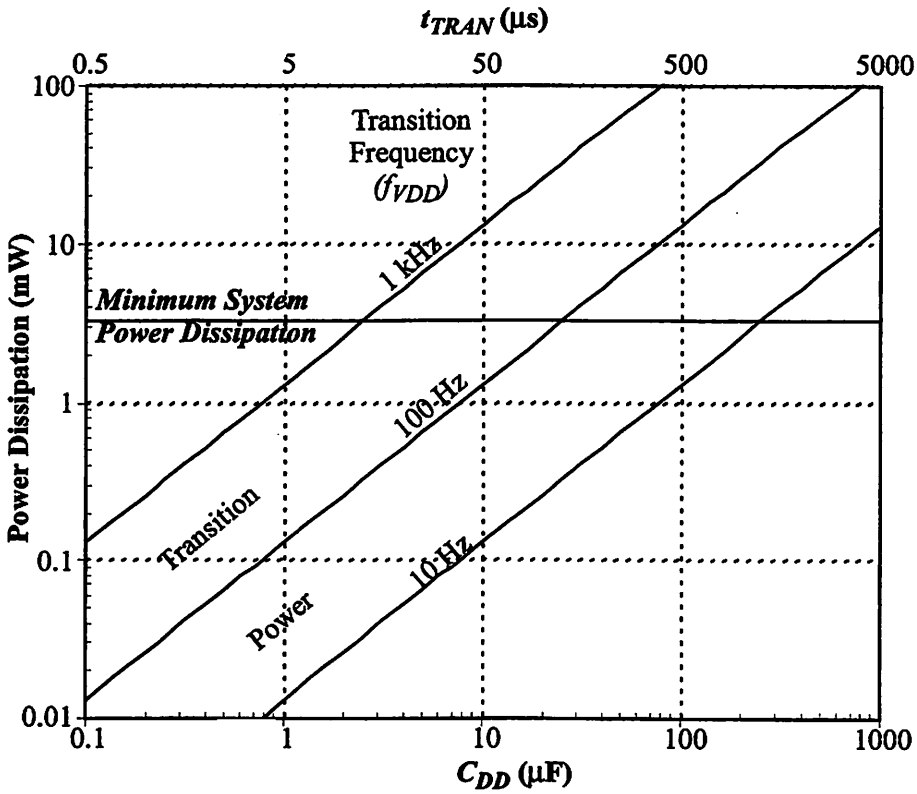


FIGURE 3.9 : Transition Time and Power Dissipation vs. C_{DD}.

combined with the hybrid PWM/PFM algorithm, allowed a dynamic voltage regulator to be designed which maintains good conversion efficiency at much lower values of C_{DD}.

3.2.8 Limits to Reducing C_{DD}

Decreasing C_{DD} reduces transition time, and by doing so increases the speed at which the voltage changes, dV_{DD}/dt. CMOS circuits can operate with a varying V_{DD}, but only up to a point, which is process dependent. This is discussed in further detail in Section 3.4.

Decreasing C_{DD} increases supply ripple, which in turn increases processor energy consumption as shown in Figure 3.10. The increase is moderate at high V_{DD}, but begins to increase as V_{DD} approaches V_T because the negative ripple slows down the processor so much that most of the computation is performed during the positive ripple, which decreases energy efficiency. For values of supply ripple above 10%, the

processor can still operate properly, but the increased energy consumption of the processor outweighs the decreased transition energy consumption, degrading overall system energy-efficiency.

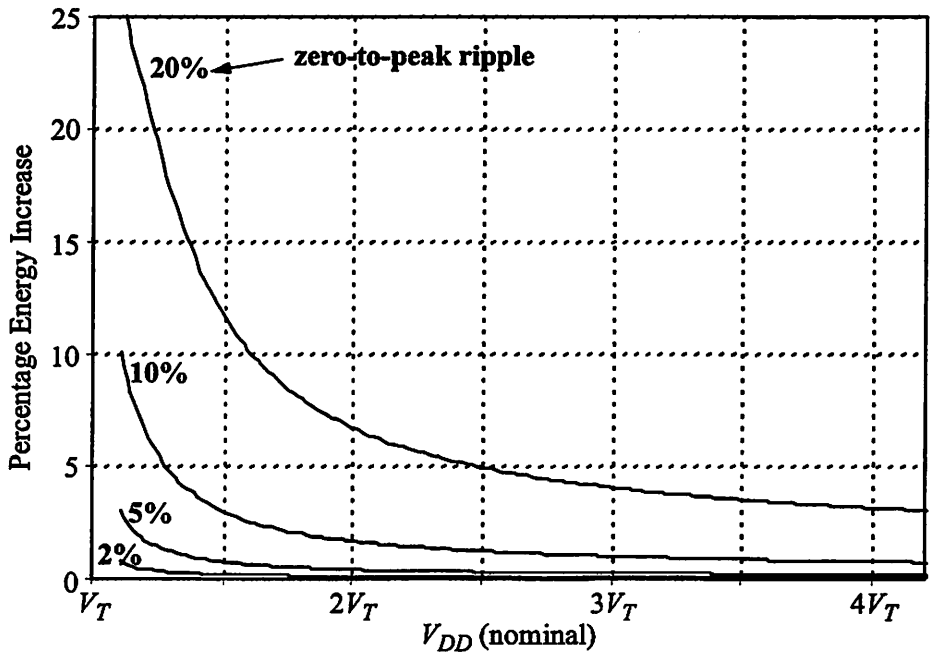


FIGURE 3.10 : Energy Loss Due to Voltage Supply Ripple.

Loop stability is another limitation on reducing capacitance. As described in Section 3.2.3, the dominant pole in the system is inversely proportional to C_{DD}. As C_{DD} is reduced the pole frequency increases. As the pole approaches the sampling frequency, interaction with higher-order poles will eventually make the system unstable.

The third limitation is that low-voltage conversion efficiency scales down with C_{DD}. Since the DVS processor will ideally be operating most of the time at low voltage, it is important to maintain reasonable low-voltage conversion efficiency.

Increasing the converter sampling frequency will reduce the supply ripple and increase the pole frequency due to the sample delay. Thus, these two limits are not fixed, but can be varied. However, increasing the sampling frequency has two negative side-effects. First, low-load converter efficiency will decrease because the converter loop will need to be activated more frequently to maintain the same voltage. Second,

the f_{CLK} quantization error will increase. These side-effects may be mitigated with a variable sampling frequency that adapts to the system power requirements (e.g. V_{DD} and I_{DD}).

The maximum dV_{DD}/dt at which the circuits will still operate properly is a hard constraint because system failure can be induced, but occurs for a much smaller C_{DD} than the supply ripple and stability constraints. Low-voltage conversion efficiency is a soft-constraint, but cannot be improved by adjusting the converter sampling frequency.

3.2.9 Optimizing C_{DD} in the Prototype System

For the prototype system, a value of 5μF was chosen for C_{DD}. The limiting factor for not reducing it further was the low-voltage conversion efficiency. Table 3.1 lists the key converter performance parameters for both the typical value of C_{DD} (100μF) and the optimized value (5μF). The top four parameters were optimized given the three bottom hard constraints.

TABLE 3.1 Converter Performance Parameters

Parameter	Constraint	C_{DD} = 100μF	C_{DD} = 5μF
E_{TRAN}	minimize	130 μJ/transition	6.5 μJ/transition
t_{TRAN}	minimize	~520 μs	~26 μs
η (3.3V)	maximize	> 95%	92%
η (1.2V)	maximize	> 95%	84%
ripple	< 10%	< 1%	2%
dom. pole	< 100 kHz	400 Hz	7 kHz
dV_{DD}/dt	< 5 V/μs	0.01 V/μs	0.2 V/μs

The optimized value maintains the constraints placed on supply ripple, the dominate pole frequency, and dV_{DD}/dt, while minimizing E_{TRAN} and t_{TRAN} and maintaining good high voltage (3.3V) and low-voltage (1.2V) conversion efficiency. There is still plenty of margin for the hard constraints, which would allow for an even

smaller C_{DD} if the converter loop could be redesigned to compensate for the reduction in low-voltage conversion efficiency, and continue to maintain a reasonable value (> 80%).

3.3 Design Constraints Over Voltage

A typical processor targets a fixed supply voltage, and is designed for ±10% maximum voltage variation. Correct functional operation must be verified over this small voltage variation, as well as a slew of timing constraints, such as maximum path delay, minimum path delay, maximum clock skew, and minimum noise margin. In contrast, a DVS processor must be designed to operate over a much wider range of supply voltages, which impacts both design implementation and verification time.

3.3.1 Circuit Design Constraints

To realize the full range of DVS energy efficiency, only circuits that can operate all the way down to V_T should be used. NMOS pass gates are often used in low-power design due to their small area and input capacitance. However, they are limited by not being able to pass a voltage greater than V_{DD} - V_{Tn}, such that a minimum V_{DD} of $2 \cdot V_T$ is required for proper operation. Since throughput and energy consumption vary by 4x over the voltage range V_T to $2 \cdot V_T$, using NMOS pass gates restricts the range of operation by a significant amount, and are not worth the moderate improvement in energy efficiency. Instead, CMOS pass gates, or an alternate logic style, should be utilized to realize the full voltage range of DVS.

As previously demonstrated in Figure 3.1, the delay of CMOS circuits track over voltage such that functional verification is only required at one operating voltage. The one possible exception is any self-timed circuit, which is a common technique to reduce energy consumption in memory arrays. If the self-timed path layout exactly mimics that of the circuit delay path as was done in the prototype design, then the paths

will scale similarly with voltage and eliminate the need to functionally verify over the entire range of operating voltages.

3.3.2 Circuit Delay Variation

While circuit delay tracks well over voltage, subtle delay variations exist and do impact circuit timing. To demonstrate this, three chains of inverters were simulated whose loads were dominated by gate, interconnect, and diffusion capacitance respectively. To mimic paths dominated by stacked devices, a fourth inverter chain was simulated in which both the PMOS and NMOS transistors were each source-degenerated with an additional three equally-sized series transistors, effectively modeling a four-transistor stack. The relative delay variation of these circuits is shown in Figure 3.11 for which the baseline reference is an inverter chain with a balanced load capacitance similar to the ring oscillator.

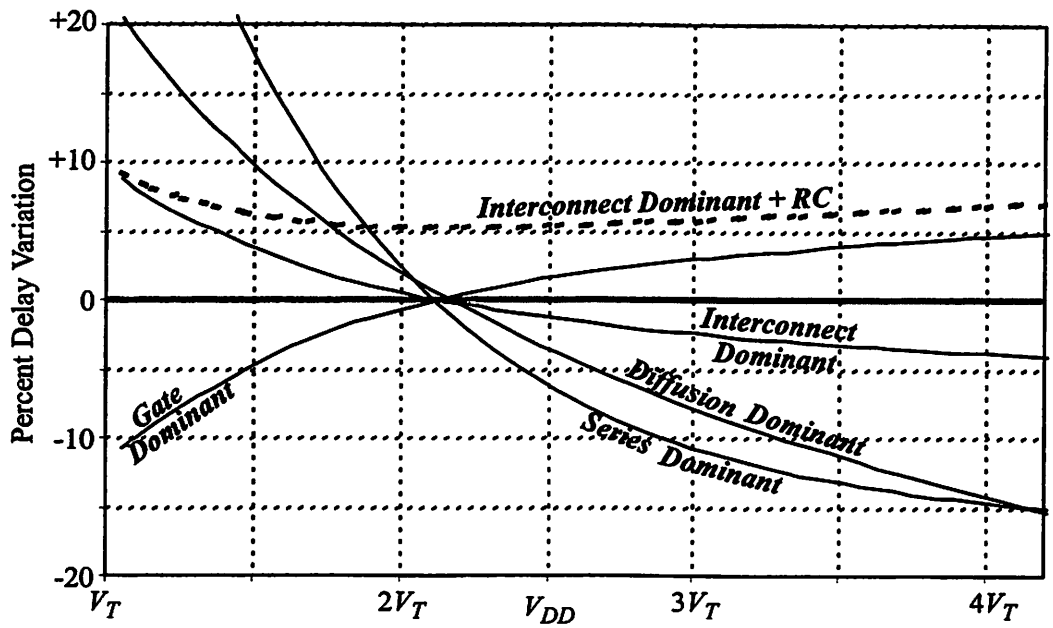


FIGURE 3.11 : Relative CMOS Circuit Delay Variation over Supply Voltage.

The relative delay of all four circuits is a maximum at only the lowest or highest operating voltages. This is true even including the effect of the interconnect's RC delay. Since the gate dominant curve is convex, combining it with one or more of

the other effects' curves may lead to a relative delay maxima somewhere between the two voltage extremes. However, all the other curves are concave and roughly mirror the gate dominant curve such that this maxima will be less than a few percent higher than at either the lowest or highest voltage, and therefore insignificant. Thus, timing analysis is only required at the two voltage extremes, and not at all the intermediate voltage values.

As demonstrated by the series dominant curve, the relative delay of four stacked devices rapidly increases at low voltage. Additional devices in series will lead to an even greater increase in relative delay. As supply voltage increases, the drain-to-source voltage increases for the stacked devices during an output transition. For the devices whose sources are not connected to V_{DD} or ground, their body-effect increases with supply voltage, such that it would be expected that the relative delay would be a maximum at high voltage. However, the sensitivity of device current and circuit delay to gate-to-source voltage exponentially increases as supply voltage goes down. So even though the magnitude change in gate-to-source voltage during an output transition scales with supply voltage, the exponential increase in sensitivity dominates such that stacked devices have maximum relative delay at the lowest voltage.

Thus, to improve the tracking of circuit delay over voltage, a general design guideline is to limit the number of stacked devices, which was four in the case of the prototype design. One exception to the rule is for circuits in non-critical paths, which can tolerate a broader variation in relative delay. By using the clocking methodology described in Section 4.3, it can be ensured that this broader variation does not lead to potential race conditions. Another exception is for circuits whose alternative design would be significantly more expensive in area and/or power (e.g. memory address decoder), but the circuits must still be designed to meet timing constraints at low voltage.

3.3.3 Noise Margin Variation

Figure 3.12 demonstrates the two primary ways that noise margin is degraded. The first is capacitive coupling between an aggressor signal wire that is switching and an adjacent victim wire. When the aggressor and victim signals have the same logic level, and the aggressor transitions between logic states, the victim signal can also incur a voltage change. If this change is greater than the noise margin, the victim signal will glitch and potentially lead to functional failure. Supply bounce is induced by switching current spikes on the power distribution network, which has resistive and inductive losses. If the gate's output signal is the same voltage as the supply that is bouncing, the voltage spike transfers directly to the output signal. Again, if this voltage spike is greater than the noise margin, glitching, and potentially functional failure, will occur.

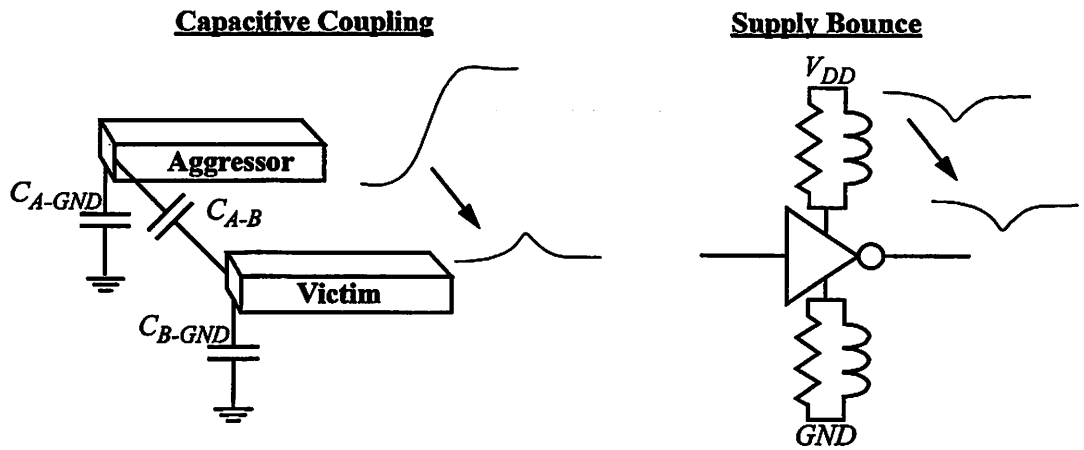


FIGURE 3.12 : Noise Margin Degradation.

For the case of capacitive coupling, the amplitude of the voltage spike on the victim signal is proportional to V_{DD} to first order. As such, the important parameter to analyze is noise margin divided by V_{DD} to normalize out the dependence on V_{DD}. Figure 3.13 plots two common measures of noise margin vs. V_{DD}, the noise margin of a standard CMOS inverter, and a more pessimistic measure of noise margin, V_T. The relative noise margin is a minimum at high voltage, such that signal integrity analysis to ensure there is no glitching only needs to consider a single value of V_{DD}. If a circuit

passes signal integrity analysis at maximum V_{DD}, it is guaranteed to pass at all other values of V_{DD}.

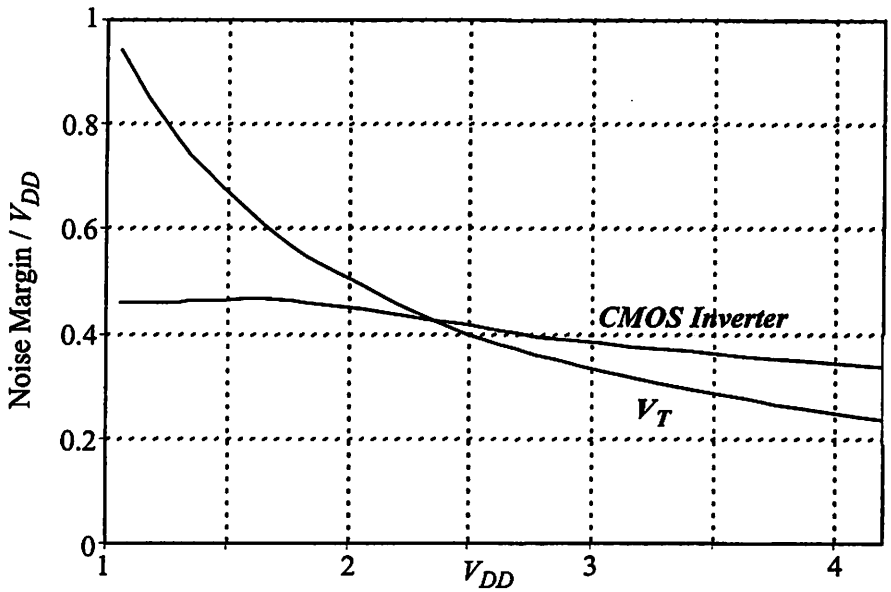


FIGURE 3.13 : Noise Margin vs. Supply Voltage.

Supply bounce occurs through resistive (*IR*) and inductive (*dI/dt*) voltage drop on the power distribution network both on chip and through the package pins. Figure 3.14 plots the relative normalized *IR* and *dI/dt* voltage drops as a function of

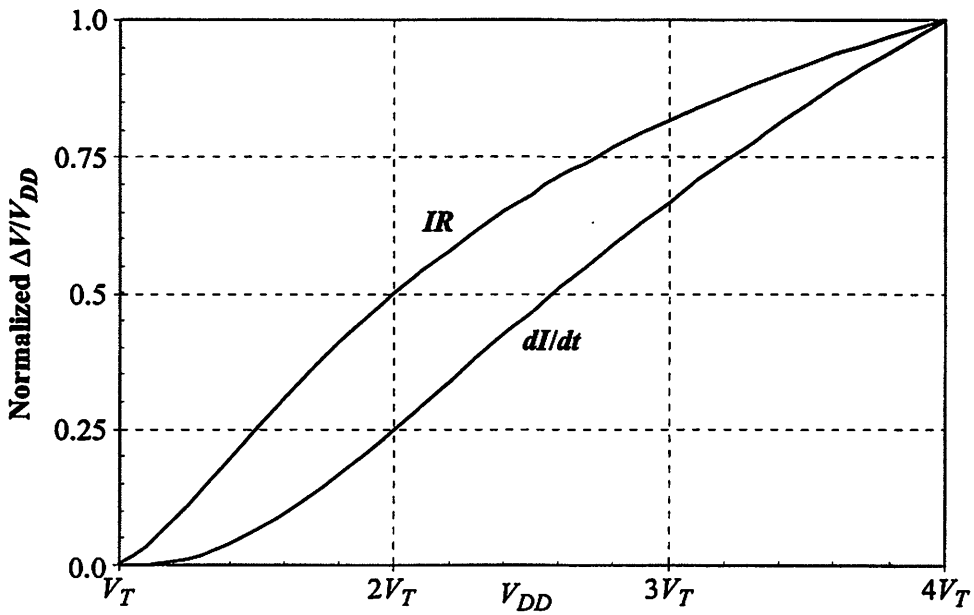


FIGURE 3.14 : Normalized Noise Margin Reduction due to Supply Bounce.

V_{DD}. It is interesting to note that the worst case condition occurs at high voltage, and not at low voltage, since the decrease in current and *dI/dt* more than offsets the reduced voltage swing. Given a maximum tolerable noise margin reduction, only one operating voltage needs to be considered, which is maximum V_{DD}, to determine the maximum allowed resistance and inductance. The global power grid and package must then be designed to meet these constraints on resistance and inductance.

3.3.4 Delay Sensitivity

Supply bounce has another adverse affect on circuit performance in that it can induce timing violations. Supply bounce decreases a transistor's gate drive, which in turn increases the circuit delay. If this increase occurs within a critical path, a timing violation may result leading to functional failure.

A typical microprocessor uses a phase-locked loop to generate a clock frequency which is locked to an external reference frequency and independent of on-chip voltage variation. As such, both global and local voltage variation can lead to timing violations if the voltage drops a sufficient amount to increase the critical paths' delay past the clock cycle time. However, in the DVS system, the clock signal is derived from a ring oscillator whose output frequency is strictly a function of V_{DD}, and not an external reference. As such, global voltage variations not only slow down the critical paths, but the clock frequency as well such that the processor will continue operating properly.

Localized supply variation, however, may only effect the critical paths, and not the ring oscillator. These can lead to timing violations if the local supply drop is sufficiently large. As such, careful attention has to be paid to the local supply routing. For the prototype design, a design margin of 5% was included in the timing verification to allow for localized voltage drops.

Delay sensitivity is the relative change in delay given a drop in V_{DD}, and can be calculated as:

$$\frac{\partial Delay}{Delay}(V_{DD}) = \frac{\partial Delay}{\partial V_{DD}} \cdot \lim_{\Delta V_{DD} \to 0}\left(\frac{\Delta V_{DD}}{Delay(V_{DD})}\right) \qquad \text{(EQ 3.9)}$$

This equation can be analytically quantified using Equation 2.13, and the normalized delay sensitivity is plotted as a function of V_{DD} in Figure 3.15. For sub-micron CMOS processes, the delay sensitivity peaks at approximately $2 \cdot V_T$. Thus, the design of the local power grid only needs to consider one value of V_{DD}, $2 \cdot V_T$, to ensure that the resistance/inductance voltage drop meets the design margin on delay variation. If the power grid meets timing constraints at this value of V_{DD}, it is guaranteed to do so at all other voltages.

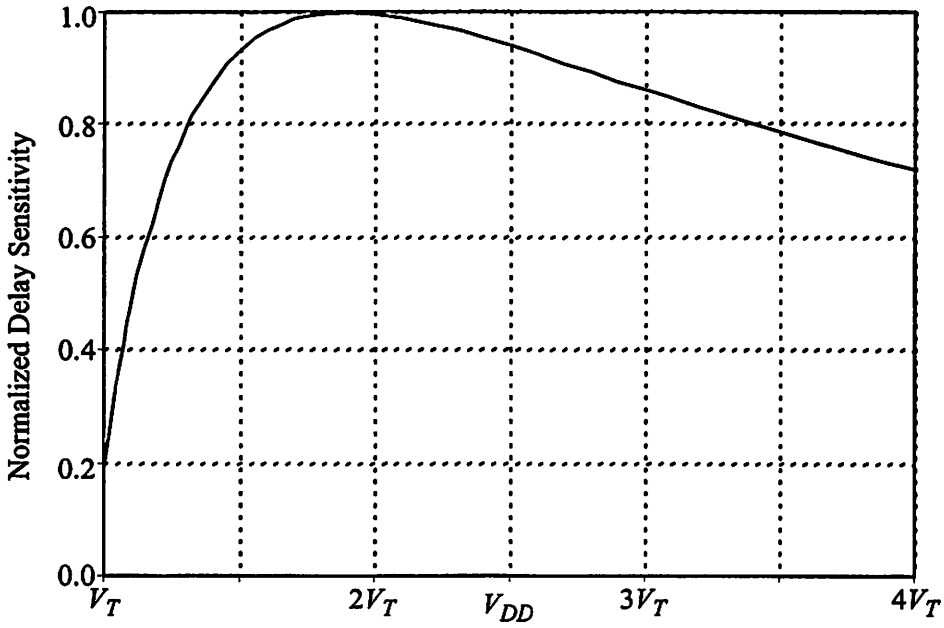


FIGURE 3.15 : Normalized Delay Sensitivity vs. Supply Voltage.

3.3.5 Summary

The verification complexity and design margins of a DVS-compatible processor are very similar to any other high-performance processor. One added constraint is that the circuits must be able to properly operate from V_T to maximum V_{DD}. Additionally, timing verification is required at both maximum and minimum

voltage, instead of just a single voltage.

Since the clock frequency is generated from the on-chip ring oscillator, the constraints on the global power distribution are actually not as severe, because only local voltage drops can induce timing failure, not global voltage drops. The only constraint put upon the global power distribution is to ensure that noise margins are met, which is much less restrictive than designing it to be immune to timing failure with an externally referenced clock signal.

3.4 Design Constraints for Varying Voltage

One approach for designing a processor system that switches voltage dynamically is to halt processor operation during the switching transient. The drawback to this approach is that interrupt latency is increased and potentially useful processor cycles are discarded. However, static CMOS gates are quite tolerable to supply voltage slew, so there is no fundamental need to halt operation during the transient.

For the simple inverter in Figure 3.16, when V_{in} is high the output remains low irrespective of V_{DD}. However, when V_{in} is low, the output will track V_{DD} via the PMOS device, and can be modeled as a simple RC network. In our 0.6μm process, the RC time constant is a maximum of 5ns, at low voltage where it is largest. Thus, the inverter tracks quite well for a dV_{DD}/dt in excess of 200 V/μs.

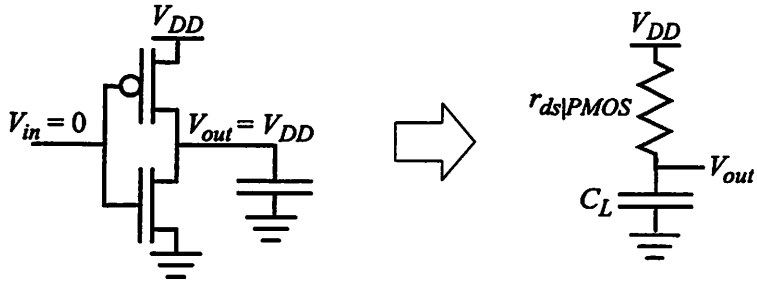


FIGURE 3.16 : Equivalent RC Network for Static CMOS Inverter.

Because all the logic high nodes will track V_{DD} very closely, the circuit delay

will instantaneously adapt to the varying supply voltage. Since the processor clock is derived from a ring oscillator also powered by V_{DD}, its output frequency will dynamically adapt as well, as demonstrated in Figure 3.17.

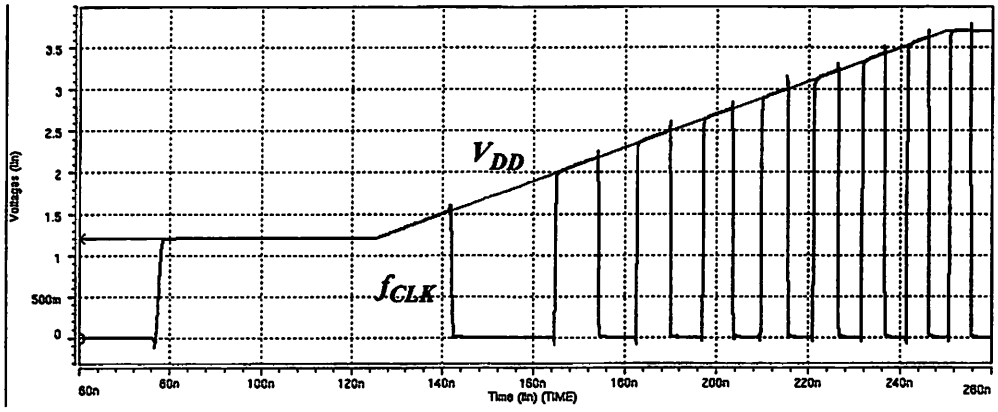


FIGURE 3.17 : Ring Oscillator Transient Performance.

Thus, static CMOS is well-suited to continue operating during voltage transients. However, there are design constraints when using a design style other than static CMOS.

3.4.1 Dynamic Circuits

Dynamic logic styles are often preferable over static CMOS as they are more efficient for implementing complex logic functions. They can be used with a varying supply voltage, as long as their failure modes are avoided by design. These two failure modes for a simple dynamic circuit are shown in Figure 3.18, and occur while the circuit is in the evaluation state (ϕ=1) and V_{in} is low. In this state, V_{out} has been precharged high, and is floating during the evaluation state.

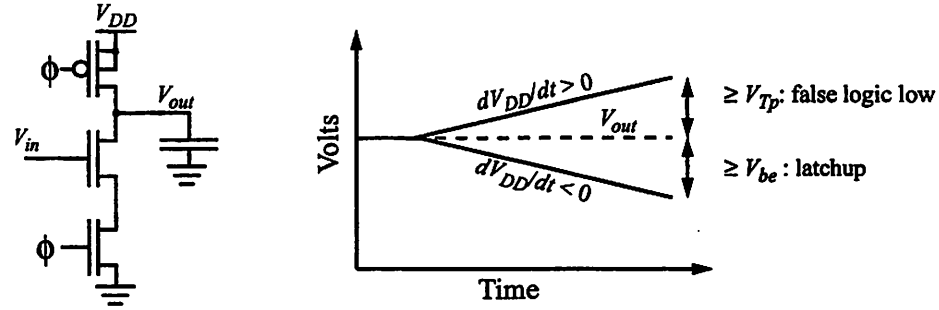


FIGURE 3.18 : Failure Modes for Dynamic Logic with Varying V_{DD}.

If V_{DD} ramps down by more than a diode drop, V_{be}, by the end of the evaluation state, the drain-well diode will become forward biased. This current may be injected into the parasitic PNP of the PMOS device and induce latchup, which leads to catastrophic failure by short-circuiting V_{DD} to ground [west93]. This condition occurs:

$$\frac{dV_{DD}}{dt} \leq \frac{-V_{BE}}{\tau_{CLK|AVE}/2} \qquad \text{(EQ 3.10)}$$

where $\tau_{CLK|AVE}$ is the average clock period as V_{DD} varies from V_{out} to V_{out} - V_{be}. Since the clock period is longest at lowest voltage, this is evaluated as V_{DD} ranges from $V_{MIN} + V_{be}$ to V_{MIN}, where $V_{MIN} = V_T + 100\text{mV}$. For our 0.6µm process, the limit is -20 V/µs, which will increase with process technology.

If V_{DD} ramps up by more than V_{Tp} by the end of the evaluation state, and V_{out} drives a PMOS device, a false logic low may be registered, giving a functional error. This condition occurs:

$$\frac{dV_{DD}}{dt} \geq \frac{V_{Tp}}{\tau_{CLK|AVE}/2} \qquad \text{(EQ 3.11)}$$

evaluated for $\tau_{CLK|AVE}$ as V_{DD} varies from V_{MIN} to $V_{MIN} + V_{Tp}$. For our 0.6µm process, the limit is 24 V/µs, which will also increase with process technology because clock frequency improvement generally outpaces threshold voltage reduction.

These limits assume that the circuit is in the evaluation state for no longer than half the clock period. If the clock is gated, leaving the circuit in the evaluation state, these limits drop significantly. Hence, the clock should only be gated when the circuit is in the precharge state.

These limits may be increased to that of static CMOS logic using a small bleeder PMOS device, as shown in Figure 3.19. The left circuit can be used in logic styles without an output buffer (e.g. NP Domino), but has the penalty of static power dissipation. The right circuit is more preferable, as it eliminates static power dissipation, and only requires a single additional device in logic styles with an output

buffer (e.g. Domino, CVSL). Since the bleeder device can be made quite small, there is insignificant degradation of performance due to the PMOS bleeder fighting the NMOS pull-down devices.

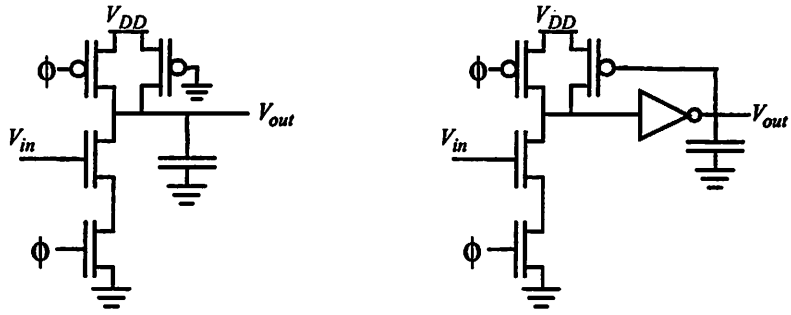


FIGURE 3.19 : Using Bleeder Devices to Improve Robustness over Varying V_{DD}.

3.4.2 Tri-state Busses

Tri-state busses that are not constantly driven for any given cycle suffer from the same two failure modes as seen in dynamic logic circuits due to their floating capacitance. The resulting dV_{DD}/dt can be much lower if the number of consecutive cycles in which the bus remains floating is unbounded. Tri-state busses can only be used if one of two design methods are followed.

The first method is to ensure by design that the bus will always be driven. This is done easily on a tri-state bus with only two drivers. The enable signal of one driver is simply inverted to create the enable signal for the other driver. However, this becomes expensive to ensure by design for a large number of drivers, N, which require routing N enable signals.

The second method is to use small cross-coupled inverters in order to continuously maintain state on the bus. This is more preferable to just a bleeder PMOS as it will also maintain a low voltage on the floating bus. Otherwise, leakage current may drive the bus high while it is floating for an indefinite number of cycles. The size of these inverters can be quite small, even for large busses. For our 0.6μm process, an inverter can readily tolerate a dV_{DD}/dt in excess of 75 V/μs with minimal impact on

performance, and only a 10% increase in energy consumption.

3.4.3 SRAM

SRAM is an essential component of a processor. It is found in the processor's cache, translation look-aside buffer (TLB), and possibly in the register file(s), prefetch buffer, branch-target buffer, and write buffer. Since all these memories operate at the processor's clock speed, fast response time is critical, which demands the use of a sense-amp. The static and dynamic CMOS logic portions (e.g. address decoder, word-line driver, etc.) of the memory respond to a changing supply voltage similar to the ring oscillator, as desired. The sense-amp, however, must be carefully designed to scale in a similar fashion.

The basic SRAM cell is shown in Figure 3.20. *Bit* and $\overline{Bit}$ are precharged to the V_{DD} value at the end of the precharge cycle. While in the precharge state, both *Bit* and $\overline{Bit}$ will track any variations on V_{DD}. Once the *Word* signal has been activated to sense the cell, the precharge devices are disabled and *Bit* and $\overline{Bit}$ do not respond to V_{DD} variations. However, the current drawn through the pass device connecting $\overline{m}$ to $\overline{Bit}$ will track V_{DD} since *Word* tracks V_{DD}, creating a voltage differential across *Bit* and $\overline{Bit}$ which will also track V_{DD}. So, to first order, the current drawn by the SRAM cell, which is a measure of its delay, will track V_{DD} similar to the delay of static CMOS logic.

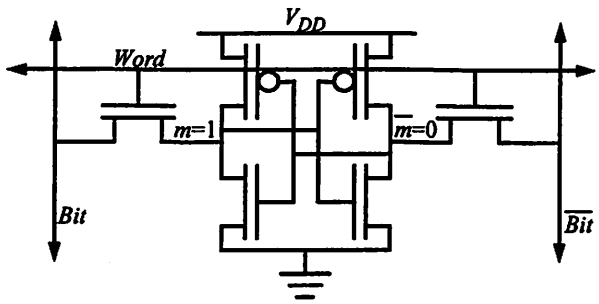


FIGURE 3.20 : SRAM Cell.

If V_{DD} drops, *m* will drop, but since *Word* will also drop, there is no affect on *Bit* since the pass device is in the off state. When V_{DD} increases, *m* will increase, as will *Word*, but will have no affect until V_{DD} increases by V_{Tn}, which is required to turn on

the pass device. When this occurs, this second-order effect will cause the voltage differential on *Bit* and $\overline{Bit}$ to increase faster because the pass device will begin charging up *Bit*, while $\overline{Bit}$ continues to be discharged. However, a dV_{DD}/dt in excess of 50 V/µs is required to induce this effect. Another second-order effect on the current drawn is that since *Bit* and $\overline{Bit}$ do not vary in the evaluation state with V_{DD}, the V_{ds} of the pass device remains constant, independent of any change on V_{DD}. However, this effect also requires large dV_{DD}/dt in excess of 50 V/µs to have any appreciable effect.

The basic sense-amp topology, shown in Figure 3.21, responds to the varying V_{DD} to first-order similar to static CMOS logic. Since the SRAM cell generates a voltage differential on *Bit* and $\overline{Bit}$ which scales with V_{DD}, the amount of time to generate the critical ΔBit to trip the sense-amp also scales.

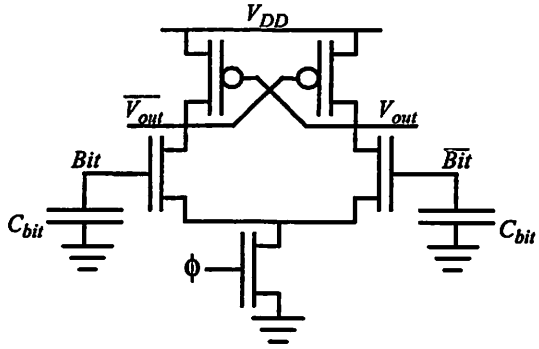


FIGURE 3.21 : Basic Sense-amp Topology.

If the common-mode voltage between *Bit* and $\overline{Bit}$ were to scale with V_{DD} as it varies during the sense-amp evaluation state, then the delay of the sense-amp would scale much like static CMOS logic. However, this is not the case, and introduces the limiting second-order effect. As V_{DD} increases, the critical ΔBit to trip the sense-amp decreases, speeding the sense-amp up. As V_{DD} decreases, the critical ΔBit to trip the sense-amp increases, slowing the sense-amp down.

Figure 3.22 plots the relative delay variation of the sense-amp compared against the relative delay variation for static CMOS for different rates of change on V_{DD}. It demonstrates that the delay does shift to first order, but that for negative

dV_{DD}/dt, the sense-amp slows down at a faster rate than static CMOS. For the prototype design, the sense-amp delay was approximately 25% of the cycle time. The critical path containing the sense-amp was designed with a delay margin of 10%, such that the maximum increase in relative delay of the sense-amp as compared to static CMOS that could be tolerated was 40%.

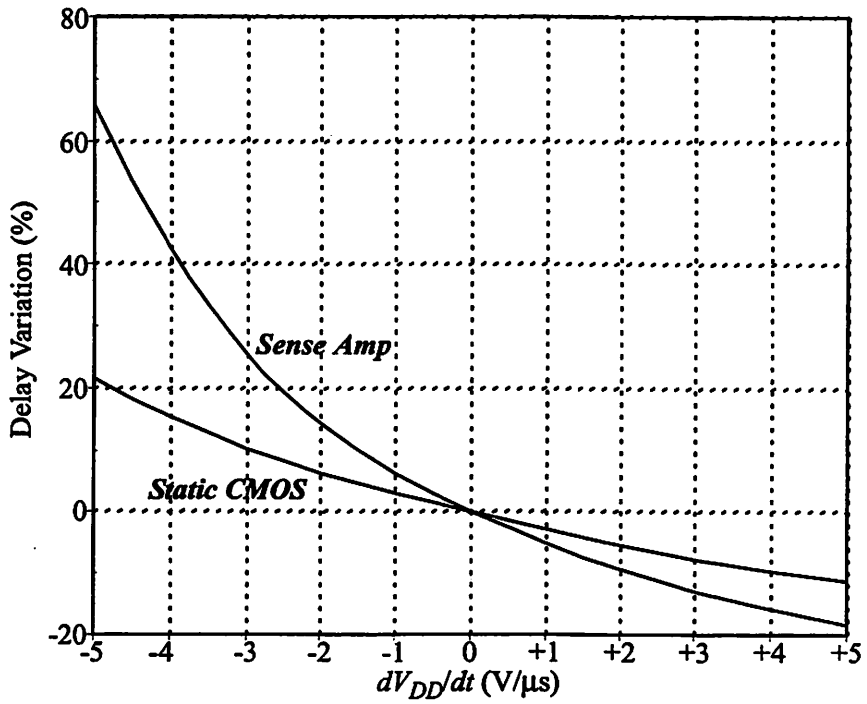


FIGURE 3.22 : Sense-Amp Delay Variation with Varying Supply Voltage.

This set the ultimate limit on how fast V_{DD} could vary in our 0.6µm process:

$$|dV_{DD}/dt| \leq 5V/\mu s \qquad \text{(EQ 3.12)}$$

This limit is proportional to the sense-amp delay, such that for improved process technology and faster cycle times, this limit will improve.

What must be avoided are more complex sense-amps whose aim is to improve response time and/or lower energy consumption for a fixed V_{DD}, but fail for varying V_{DD}. One example is a charge-transfer sense-amp [burs97][kawa98].

3.4.4 Summary

As was demonstrated for the sense-amp, simpler circuit design ensures greater DVS compatibility. Many circuit design techniques developed for low power, such as the charge-transfer sense-amp and NMOS pass-gate logic, are not amenable to DVS. However, the potential energy efficiency improvement of DVS far outweighs the slight degradation in energy efficiency by not using these more energy-efficient circuit design techniques.

In addition, a methodical design approach must ensure that no signal is ever floating for more than a half-cycle to prevent functional errors. But even with this approach, there are limits to dV_{DD}/dt, on the order of 20 V/µs for our 0.6µm process. Higher dV_{DD}/dt can be tolerated for dynamic circuits with the use of bleeder and feedback devices, but is not required since the sense-amp is the limiting factor. While the basic sense-amp's delay scales to first-order with dV_{DD}/dt, second-order effects limit $|dV_{DD}/dt|$ to only 5 V/µs in our 0.6µm process. However, this limit will increase with better CMOS process technology.

3.5 Voltage Scheduler

To realize the full benefit of DVS, not only must the hardware support operation at any desired voltage level, but the operating system must be intelligent enough to set the processor speed based upon the current workload, and estimated future workload requirements. A new element has been added to the operating system to perform this new functionality, which is called the voltage scheduler.

A more comprehensive description of the voltage scheduler can be found in [peri00]. This section only describes it in brief to highlight the key components and functionality.

3.5.1 Application Execution Models

Evaluating workload requires knowledge of an application's allowable execution time, and the number of instructions that need to be executed in that time frame. Based upon execution time requirements, software applications fall into one of three general classifications: deadline-based tasks, rate-based tasks, and high-priority tasks.

Deadline-based tasks are applications that can be clearly divided into execution units called frames, each of which has a measure of work which should be completed by its deadline [burn97]. A frame is an application-specific unit, such as a video frame, and work is defined in terms of processor cycles. The desired rate of processor execution in a single-threaded system is easily determined:

$$Processor\ Speed = \frac{Processor\ Cycles}{Deadline} \qquad \text{(EQ 3.13)}$$

It is the job of the voltage scheduler to determine the processor speed given a set of multiple tasks. The work required is automatically estimated by the system and updated to reflect the actual work performed as the application executes. The applications must provide some level of buffering to accommodate missed deadlines.

Rate-based tasks sustain a predictable rate of execution, supplied by the application, without an explicit deadline. Applications such as compilation, which should finish in "a reasonable amount of time," fall into this category. The deadline for these tasks is automatically calculated by the internal scheduler and update dynamically as the task executes. Rate-based threads should be scheduled so that they appear to be executing on a single-threaded system running at their specified speed: two 10 MHz sustained rate threads will require time-sharing a system run at 20 MHz.

High-priority tasks are short, sporadic tasks which require quick response times but do not present a significant system load. They execute before all other threads

in the system; high-priority tasks themselves are ordered and execute highest-priority-first. High-priority tasks are ignored by the voltage scheduler and do not directly effect the processor speed.

Rate-based and high-priority tasks can be represented as deadline-based with artificial deadlines, reducing all software applications to a single execution model which can be quickly evaluated by the voltage scheduler via Equation 3.13. Applications default to a rate-based model until they specify otherwise. Most system-level threads run at high-priority, while multimedia tasks typically use a deadline-based model.

3.5.2 Workload Prediction

An accurate workload prediction (W) is necessary to allow a voltage scheduler to efficiently schedule the system. The workload predictions for deadline-based threads are determined empirically at run-time using an exponential moving average:

$$W_{NEW} = \frac{W_{OLD} \cdot k + W_{LAST}}{k+1} \qquad \text{(EQ 3.14)}$$

The operating system updates the associated estimate each time a frame is completed. An application may either explicitly specify an initial work estimate or use the computation required for its first frame to initialize the sequence. Rate-based applications specify their workload as sustained rate of execution, defined in terms of processor speed (equivalent to work/time). High-priority threads do not need to specify a workload estimate because they are ignored by the voltage scheduler.

Due to frame-to-frame application variance, the actual work required will differ from the estimate. A thread will either terminate before or after its requested deadline. Figure 3.23 shows two example of how this incorrect prediction can result in sub-optimal energy usage. Applications with high workload variance can potentially consume more energy than applications with consistent workloads; an efficient

scheduling techniques can combine multiple frames together to reduce the effective variance.

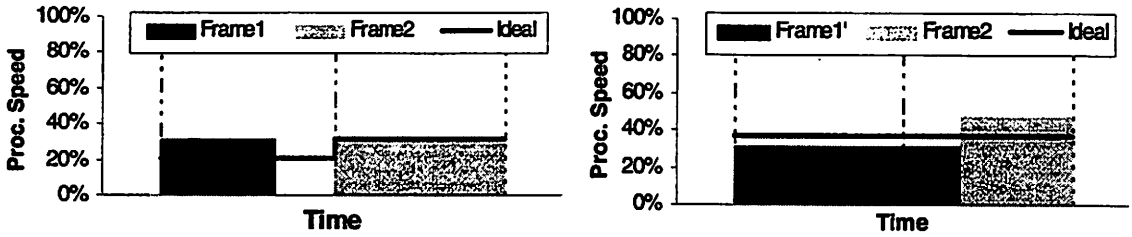


FIGURE 3.23 : Workload Over/Under Estimation

3.5.3 Integration into the Operating System

At the heart of any pre-emptive multi-tasking operating system is a temporal process scheduler. Its responsibility is to maintain the list of active and sleeping threads, and schedule time slots for them to run in. The optimal algorithm for the temporal scheduler is an earliest-deadline first (EDF) policy for a fixed-speed system [liu73].

The voltage scheduler determines for a given process schedule, what the optimal speed is to complete the outstanding threads by their specified deadlines for minimal energy consumption. Thus, the voltage scheduler can be independent of the underlying temporal scheduler, simplifying its integration into an existing operating system.

Whenever the temporal scheduler executes, it invokes the voltage scheduler with the updated process table containing the active list of threads and their corresponding deadlines. The voltage scheduler is only responsible for estimating the workload for this list of threads and calculating the optimal speed at which the processor should be operating.

3.5.4 Zero-Start Algorithm (ZERO)

The ZERO algorithm is the culmination of work investigating an optimal, yet implementable, voltage scheduler [peri00]. The basic algorithm assumes all tasks are sporadic and calculates the minimum speed necessary assuming their relative start times are all zero. Given that threads are sorted in EDF order, the speed can be found:

$$Processor\ Speed = \underset{\forall(i \le n)}{MAX}\left(\frac{\sum_{j \le i} work_j}{deadline_i - currenttime} \right) \qquad \text{(EQ 3.15)}$$

The zero-start simplification causes ZERO to overestimate the processing required when a future task has a large amount of work to be completed. Additionally, ZERO will underestimate requirements for executing multiple periodic threads since it treats all tasks as sporadic. The algorithm can be implemented in $O(n)$ time where n is the number of scheduled threads; the required list of sorted threads can be maintained incrementally as task deadlines are updated. To improve the algorithm's operation, it has an additional four extensions:

Schedule Smoothing - Threads which have exceeded their deadline or work estimate are scheduled so as to complete twice the work with twice the deadline. Without this modification, a missed deadline or work underestimate will run the processor at full speed to minimize frame completion latency (unless otherwise specified by the over-deadline policy, below). Since applications are required to tolerate missed deadlines, smoothing the schedule reduces energy without significantly affecting system behavior.

Over-Deadline Policy - A thread may opt to be switched to a rate-based model when its deadline or work estimate is exceeded. This technique is most useful for applications which have the occasional long-running frame which is not latency-critical, such as an initialization sequence. The over-deadline policy is complementary

to scheduling smoothing: over-deadline handles frames which are over their deadline/ estimate by a significant amount, while schedule smoothing handles frames which are still operating close to their deadline/estimate. When used in conjunction with schedule-smoothing, the over-deadline policy is invoked at twice the deadline and work estimate. In the absence of an over-deadline policy, the task will be executed at the maximum processor speed to minimize time-to-completion.

Extra Work - Aggregate workload statistics of all high-priority threads are accumulated and used to provide an estimate of their future work requirements. This estimate is used to reserve cycles in the processor workload for future high-priority tasks, which are otherwise not included by the voltage scheduler. This technique adjusts execution so that frames complete closer to their deadline.

Event Filtering - Frames which could not possibly complete before their deadline, even with the processor running at max-speed, are not included in the automatic workload estimation. This filtering optimizes for the useful common case. A user-interface, for example, consists of many short events, such as button refreshes, with a few lengthy events, like opening a new dialog window. If the lengthy events could never possibly complete before the stated deadline incorporating them into the work estimate would only serve to inflate the estimate, adversely affecting the common case. Threads are initially scheduled assuming a short frame, and then dynamically adjusted for longer running ones.

3.5.5 Operation

A demonstration of the voltage scheduler on the prototype processor system is shown in Figure 3.24, which plots V_{DD} over a two-second window. The top trace demonstrates system operation without the voltage scheduler; the system either operates at max speed or idles. The bottom trace demonstrates system operation with the voltage scheduler enabled, in which case V_{DD} is being dynamically varied.

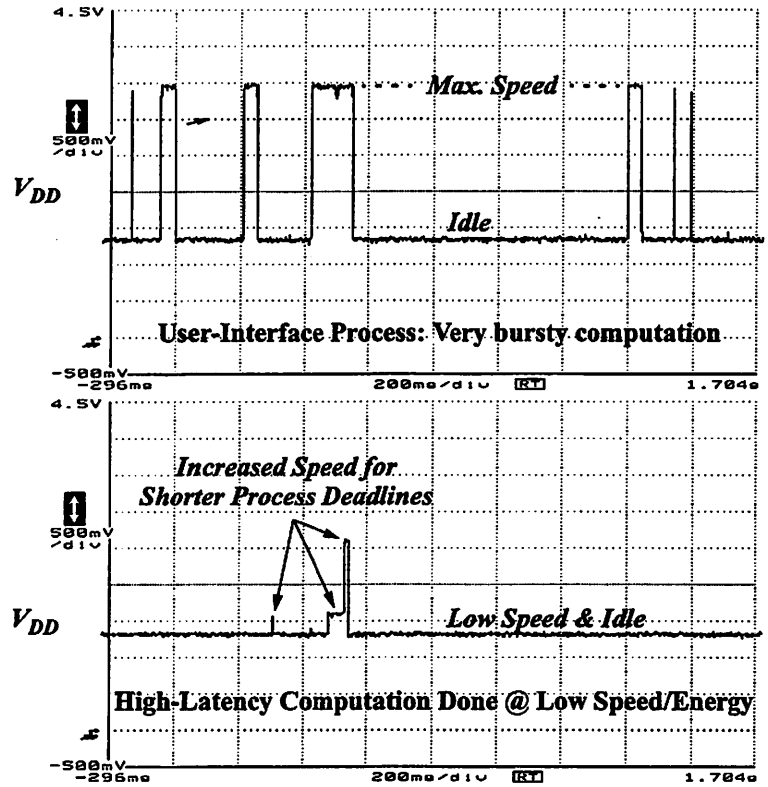


FIGURE 3.24 : Voltage Scheduler Improvement

For the user-interface application being executed, most of the processing can be done at low voltage, greatly improving energy efficiency. When a large amount of processing needs to be performed, the scheduler will ramp up V_{DD} to meet the specified deadline. If a deadline is missed, the scheduler begins ramping up the processor speed.

3.6 Benchmark Evaluation

The ideal energy efficiency improvement of DVS is more than 10x. However, to evaluate the practical improvement of DVS, it is important to have a benchmark suite that mimics code that would actually be running on the processor system.

A class of systems where DVS is expected to have enormous impact are user-driven portable electronics such as notebook computers and PDAs. While DVS is quite applicable elsewhere, the scope of this benchmark evaluation only focuses on these systems.

3.6.1 Description

Two common multimedia tasks and a graphical user interface are used to evaluate DVS, which are commonly found in this class of systems:

• **Audio** - IDEA decryption of a 10-second 11 kHz mono audio stream, divided into 1kB frames with a 93ms deadline. In a single-task environment, this benchmark runs at approximately 17 MHz.

• **MPEG** - MPEG-2 decoding of an 80-frame 192x144 video at 5 frames/sec, requiring an average of 50 MHz processing.

• **UI** - A simple address-book user interface allowing simple searching, selection, and database selection. 432 frames are processed, each defined as a user triggered event, such as pen-down, which ends when the corresponding action has been completed. A deadline of 50ms, described below, is used for each frame. Most frames require less than 10 MHz operation.

High-level wrappers are used by applications to specify real-time constraints and manage work estimates. Each iteration, periodic applications (Audio & MPEG) update their work estimate, set their next deadline based on the period, and stall until the appropriate start time. Sporadic applications (UI) specify constraints using start/ stop times. The Audio benchmark supplies an initial work estimate based on previous execution, while the MPEG and UI benchmarks use the results of their first completed frame, which is run at max-speed. The wrapper for applications with hard deadlines (Audio & MPEG) must contain small buffers to tolerate missed deadlines.

Simulated user input (mouse & keyboard) is the primary mode of input for the UI application. A 50ms deadline representing human visual perception time is used for UI events, as screen updates faster than this will not be noticed by the user [endo96]. Most UI events, such as simple button updates, can easily complete before this deadline. Some events, however, are extremely computationally expensive; opening a

new dialog window, for example, takes 260ms when the processor is running at max-speed. There is no "correct" speed for these events. To balance energy-efficiency and low-latency, a speed of 40 MHz is chosen.

3.6.2 Detailed Performance Analysis

To evaluate DVS energy efficiency improvement, the current consumed from the battery supply can be monitored over the duration of the benchmark and integrated to calculate total energy consumption with and without DVS enabled. The ratio of these two energy measurements is then the energy efficiency improvement.

However, to understand the dynamics of the voltage scheduler algorithm, and determine whether it is working as desired, more detailed monitoring was performed, as shown in Figure 3.25. At regular 10μs intervals, the operating system reports the active thread running and the speed it is running at. In this example, there are six active threads, labelled along the vertical axis. A low value indicates the thread is not running

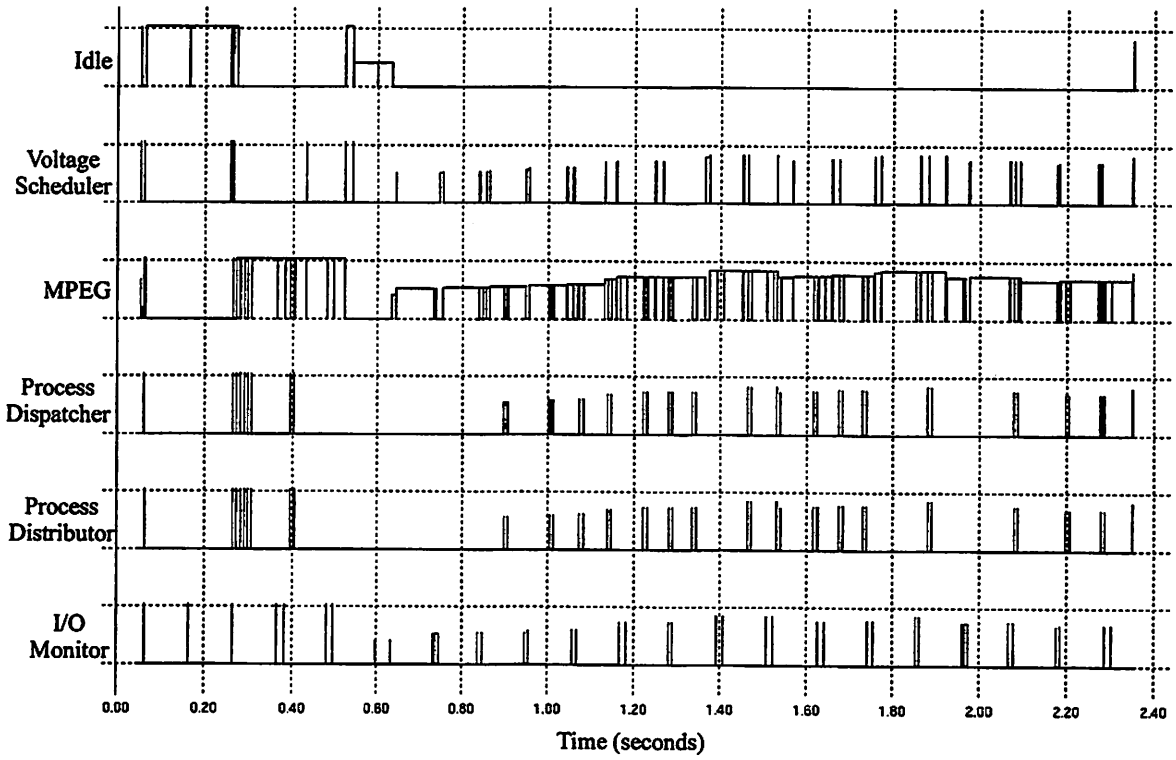


FIGURE 3.25 : Run-time Performance Analysis

at that moment. A high value indicates it is running and the amplitude is proportional to the current speed setting

The Idle thread initially dominates run-time until the MPEG benchmark thread has become initialized. Once initialized at maximum speed, the ZERO algorithm begins estimating the optimal MPEG speed setting, and adjusting it depending on the current frame workload. The I/O monitor runs at constant intervals reading in the MPEG data arriving at a fixed rate. The temporal scheduler (not shown) runs at regular intervals, maintaining the process schedule. When the process schedule is modified, the voltage scheduler, process dispatcher, and process distributor threads are launched.

With this monitoring, it can be observed that the MPEG benchmark dominates total CPU time, which is desired. Also, the speed is being varied around to adapt to the varying computational requirements of the MPEG frames. The voltage scheduler and other operating system threads only consume a fraction of the CPU performance. This performance monitoring greatly aided in the development and tuning of the ZERO voltage scheduling algorithm

4 Energy Conscious Design Flow

The most critical aspect of energy-efficient design is to be energy conscious throughout the entire design flow. A typical design flow treats energy consumption as an afterthought, and is not thoroughly analyzed until the design has reached the transistor schematic stage. This is too late in the design process for radical modifications that can lead to a more energy-efficient implementation. Therefore, much as performance is analyzed at the initial high-level specification of the design, so must energy consumption be analyzed, as well. The primary goal of this design flow is to evaluate energy consumption early on so that the largest energy reductions can be attained.

In today's complex chip designs, a majority of the design cycle is spent on validating a design for proper functionality, and verifying its layout implementation. Implementing DVS exacerbates the verification problem by requiring multiple operating conditions to be analyzed. Another goal of this design flow is to automate design validation and verification so that the bulk of the design effort could remain focused on the design implementation and optimization for energy efficiency.

The first section presents an overview of the design flow. Subsequent sections explore in more detail those parts of the design flow that were developed to aid in the design and verification of a DVS microprocessor system, though most of this design

flow is equally applicable to general energy-efficient digital design.

4.1 Overview

The basic design flow from system specification to final chip layout is presented in Figure 4.1. The flow refines the design through five discrete phases, each of which has its own set of design tools, and optimization goals. Through each refinement phase, the design is verified against the previous phase's implementation to

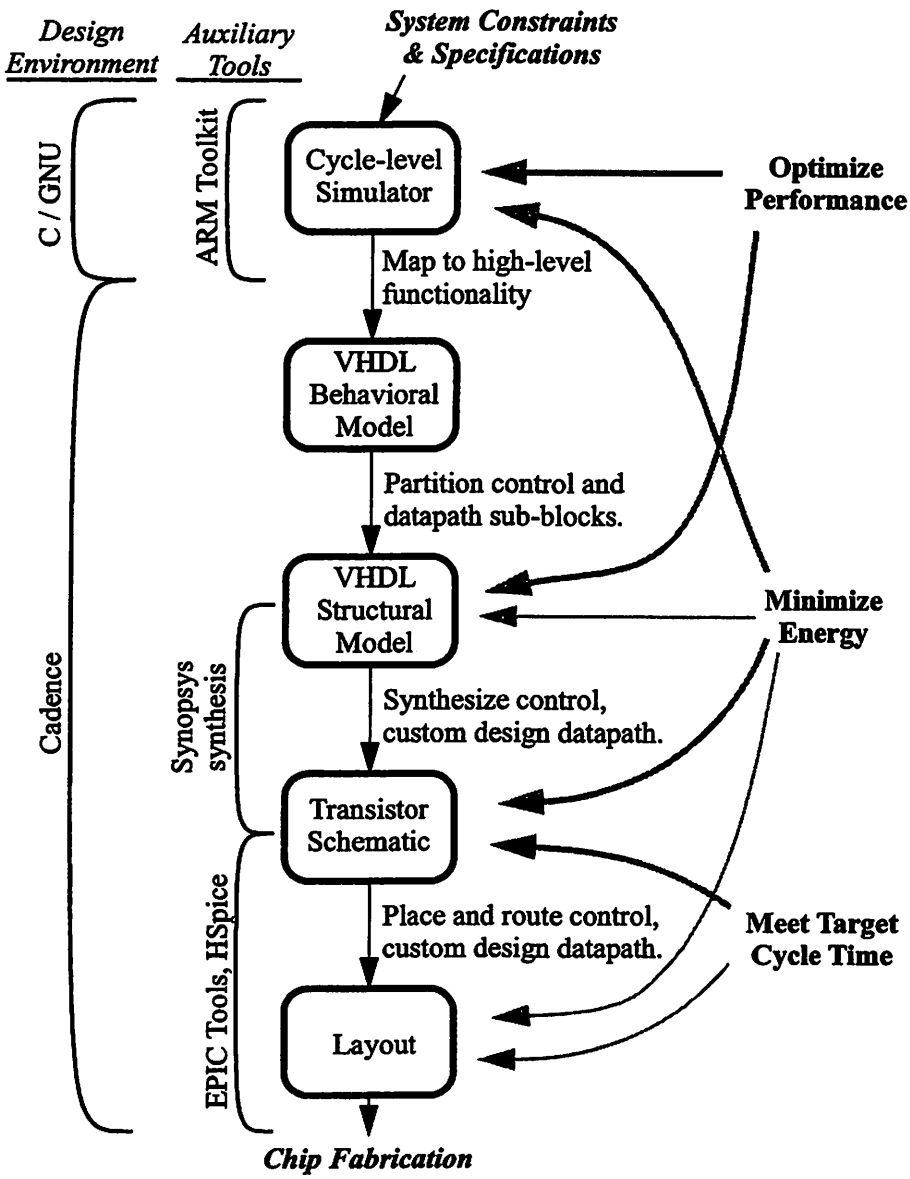


FIGURE 4.1 : General Design Flow from Specification to Layout.

insure proper functionality. Through transitive equivalence, the layout can be verified to operate as specified by the initial cycle-level simulator.

Implementing an existing instruction set architecture (in this particular case the ARM v4 ISA [arm96a]) provides the key benefit of having proven compilers and assemblers available. The design verification process is bootstrapped by the ability to swiftly write self-validating C code which can be compiled down to machine code and executed on the simulator. This allows the simulation of real test code early on with abstract design models.

An advantage of implementing the ARM V4 ISA was the availability of a simulator (from ARM Ltd.) for the processor core [arm97]; once an abstract memory/IO module was written, the simulator could begin booting the operating system and executing benchmark programs. This provided the ability for high-level performance and energy estimation so that the design could be optimized as the simulator evolved into a cycle-accurate specification. Performance tuning at this stage is common in microprocessor design, but energy estimation is typically done as an afterthought. By incorporating energy estimation into the simulator, high-level energy optimization significantly improved the design's energy efficiency as will be discussed in Section 4.2.

Once the performance and energy optimized cycle-accurate simulator was developed, the design progressed to the VHDL behavioral design phase. Since the design is still at a behavioral-level specification, no significant optimizations are possible at this level. Since the specification language has changed from C to VHDL, it is critical at this juncture that the two models behave exactly the same, as discussed in Section 4.5.1. The design was not initially specified in VHDL because C simulation is several orders of magnitude faster, which enabled a rapid design turn-around during the initial specification phase.

As the VHDL description transitions from behavioral to structural models, additional performance and energy optimization occurs. By annotating the structural VHDL model with delays, critical paths can be identified and reduced. At this microarchitecture level, better energy estimates are available for the individual blocks, and the simulator is updated accordingly to provide better energy estimates. Dominant energy consuming blocks can be identified and optimized. By the end of this phase, remaining critical paths that cannot be removed via microarchitecture changes dictate the achievable cycle time. At this point, the performance optimization transforms to meeting the targeted cycle time.

Another significant level of energy optimization occurs during the transistor design utilizing the various energy-efficient design techniques in Section 6.1. During this phase, PowerMill provides accurate energy estimates which aid the designer to further reduce energy consumption. Accurate timing analysis (Section 4.6) ensures the target cycle time can be met. Simple schematic redesigns are done when the target cycle time cannot be met, as indicated by long path lengths, which is much less costly than waiting for timing results on the extracted layout.

During the final layout phase, energy again can be minimized through intelligent layout, but to a much less degree than previous phases. Through LVS, the layout can be matched back to the schematic to verify a correct design. Timing analyses on an extracted layout netlist can flag additional critical paths that need to be fixed which were not flagged during the schematic design phase.

4.1.1 Energy Budgeting

To most effectively optimize energy consumption, energy reduction in total system energy must be measured, and not just localized energy reduction. For example, if a particular design change reduces the energy consumption of the write buffer 50% while degrading system performance by only 10%, it may appear to be a desirable

optimization. If, however, the write buffer only consumes 10% of the overall energy, then a 5% reduction in overall system energy consumption does not justify a 10% performance reduction. Thus, it is imperative to always evaluate energy and performance at the system level for potential design optimizations, and not in isolation.

Section 4.2 describes in further detail a methodology for high-level system energy estimation. This provides an estimated breakdown of the system energy consumption long before the design is taken to structural and physical implementation. This breakdown provides crucial guidance on what blocks require careful design to minimize energy, and what blocks consume negligible energy and can be rapidly designed through synthesis. This breakdown is updated and maintained throughout the design process to track where the focus should be for minimizing system energy consumption.

4.1.2 Verification Overview

Verification checks are performed at each level of the design phase, as shown in Figure 4.2. At the C & VHDL behavioral levels, the checks are strictly for functionality. Test code is created using the methodology described in Section 4.5.2, which verifies that the behavioral models match the specification for the ISA and IO expected by the programmer. Scripts automatically generate test vectors for use at the structural and transistor level design phases.

Very simple timing analysis is performed at the structural model by simulating timing-annotated VHDL models. Excessively long critical paths can be identified at this stage, ensuring an implementation is possible with the target cycle time derived at the end of this design phase. Comprehensive timing analyses are performed at the schematic and layout design phases, as described in Section 4.6.

Transistor sizing checks are done at the schematic design phase to catch grossly under/oversized clock buffers and bus drivers, based upon capacitance

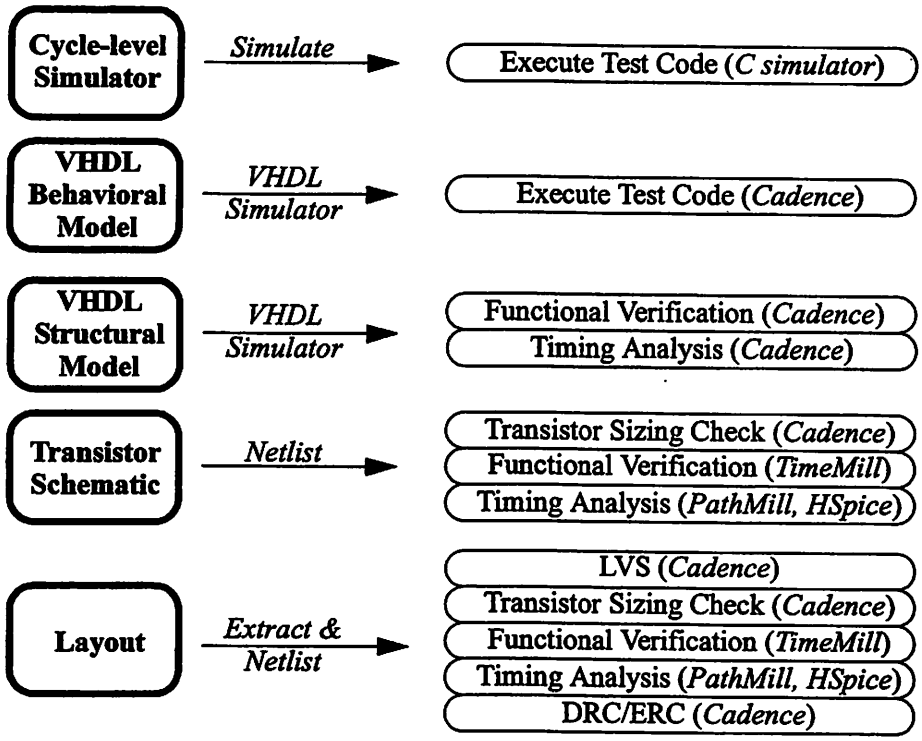


FIGURE 4.2 : Validation Checks for Each Design Phase

estimates from the schematic netlist. Once the design transitions to the layout phase, final sizing checks are performed on the extracted netlist to guarantee valid clocking operation (Section 4.3) and to minimize short-circuit current (Section 6.1.2).

Finally, the layout requires standard additional checks to verify a proper implementation (LVS, DRC, etc.). Through the use of scripts, all these tasks can be automated.

4.2 High-level Energy Estimation

High-level energy estimation for architectural exploration has been successfully demonstrated within a VHDL simulation environment [land94]. This approach is very suitable for dedicated DSP architectures. However, this is not a suitable approach for general-purpose processor systems which require several orders of magnitude more simulation cycles to properly characterize the system. A reasonable benchmark suite requires billions of simulated cycles; even on the fastest VHDL

simulator, the suite would take at least a day to complete a single pass.

To reduce simulation time, processor behavior models can be written in C, which can speed up simulation time one to two orders of magnitude, and then used to provide extensive performance statistics. However, they can also be augmented to provide energy consumption statistics as well, by applying the same black-box capacitance modeling used in the VHDL simulator [land94] to the C simulator.

This enables the high-level architectural exploration methodology shown in Figure 4.3 which has modified a standard ARM v4 processor core simulator to provide both performance and energy statistics. Initial hardware capacitance estimates are added to the simulator, which then executes the benchmark suite to provide statistics for identifying dominant energy-consuming blocks and performance bottlenecks. New optimizations are proposed to improve energy efficiency, and implemented in the C simulator, so that the improvement in energy efficiency of these design optimizations

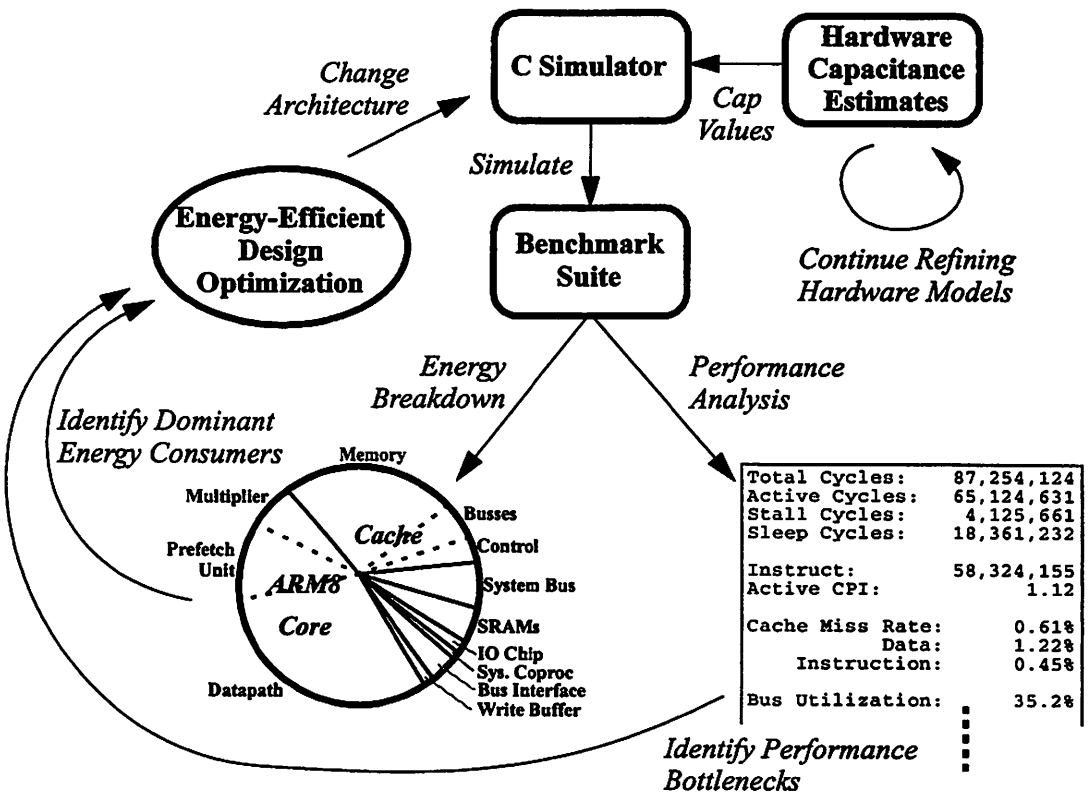


FIGURE 4.3 : High-level Simulation Performance & Energy Estimation Methodology

can be quantified after resimulating the benchmark suite.

An example simulator output report file:

```
Report date: Thu Feb 5 12:26:45 1998
Report tag: 'No ID Specified'
Report generated to ./report_fn.out

Sim time:     155.0ms
Run time:     118.0s
Simulation speed 1/761th
Running on yellowstone...

Energy Estim. for ARMulator 0 level 1
-------------------------------------
Accesses to MUL Sim Block:       8580
Accesses to ADM Sim Block:        280
Accesses to PSR Sim Block:       5963
Accesses to ALU Sim Block:    3393973
Accesses to IMM Sim Block:    1627554
Accesses to TST Sim Block:     488809
Accesses to SWI Sim Block:        623
Accesses to OTH Sim Block:          0
Accesses to SCC Sim Block:     678441
Accesses to BRA Sim Block:     587153
Accesses to EAR Sim Block:     132436
Accesses to EAI Sim Block:    1504181
Accesses to LDR Sim Block:    1424354
Accesses to STR Sim Block:     212263
Accesses to MDT Sim Block:      71870
Accesses to LDM Sim Block:      77776
Accesses to STM Sim Block:      46120
Accesses to SWP Sim Block:        465
Accesses to COP Sim Block:        762
Accesses to DEC Sim Block:    8149529
Accesses to PRE Sim Block:    9942373
Accesses to CTL Sim Block:    8148058
Accesses to CAC Sim Block:    8034207
Accesses to WBI Sim Block:     169247
Accesses to MSO Sim Block:    1170528
Accesses to SRM Sim Block:     365819
Accesses to IOC Sim Block:     304309

-------------------------------------

Block  Accesses Energy (%) Sw. Cap (%)
-----  -------- ---------- -----------
S/ALU   5663245     9.446 %     8.792 %
RegBk   6768302     0.001 %     0.001 %
Mult       8580     0.048 %     0.044 %
PSR      241694     0.134 %     0.125 %
ConGen  1040224     0.434 %     0.404 %
AddGen    65959     0.018 %     0.017 %
WrPipe   107114     0.045 %     0.042 %
CoProc      156     0.000 %     0.000 %
MemRE   1049251     0.292 %     0.271 %
Prefet  6240667     6.940 %     6.459 %
Contrl  5915641    13.156 %    12.245 %
AddMux        0     0.000 %     0.000 %
BusInt    21369     0.015 %     0.014 %
BusExt   284734     0.158 %     0.147 %
Cache   4759464    39.218 %    36.501 %
SRAM     217294     1.812 %     1.687 %
I/O        9662     0.027 %     0.025 %

Bus  Accesses  Energy    Sw. Cap   Toggle
---  -------- --------  --------  --------
A     5714811  5.224 %   4.862 %  25.687 %
B     3570615  2.591 %   2.411 %  27.185 %
RES   4433936  4.592 %   4.274 %  29.106 %
CDT       156  0.000 %   0.000 %   0.401 %
RDT  10620210 12.531 %  11.662 %  33.158 %
WDT    107114  0.116 %   0.108 %  30.465 %
VAD   7396583  3.043 %   2.832 %  11.563 %
PRO    284734  0.111 %   7.034 %  29.836 %
IO       9662  0.048 %   0.044 %  11.071 %

      Delta Energy = 0.0195862 Joules
Total Elapsed Time = 0.268321 sec
Delta Elapsed Time = 0.155443 sec
Delta Instructions = 5915641
  Removed Branches = 0
Delta Exec. Cycles = 7551141
               CPI = 1.27647
Energy / Instruct. = 3.31092 nJ
     Average Power = 126.003 mW

Instruction count report:
Data processing : 75.56%
   Set ccodes   :  5.32%
   op1 not used : 28.66%
   no result    :  4.25%
PSR Op          :  0.06%
Swap            :  0.01%
Multiply        :  0.15%
Halfword        :  0.00%*
Data transfer   : 17.63%
   No WB        : 90.07%
   Pre-incr     :  5.83%
   Post-incr    :  4.09%
   Store Byte   :  2.14%
Block Data      :  1.11%
   Stores       : 64.91%
Branch          :  2.53%
Coproc          :  0.00%
SWI             :  0.01%
Squashed        :  2.93%
---
Conditional     :  5.28%
PC Read         :  0.26%
PC Write        :  0.38%
Shifts          : 77.76%
  Src:Register  :  0.00%*
  Typ:Immediate : 25.50%
    Left 0      : 25.87%
    Left 1      :  0.39%
    Left 2      : 16.84%
    Left 3      :  0.23%
    L. others   :  6.46%
    Others      : 24.71%
5915k instructions total profiled.

Cache/Memory report:
Unified: 16kb/8w-lines/4-way/Lru
FCLK: 50MHz   MCLK: 25MHz   WB: 8g   NCW_NWB

        __hits__   __miss__  _rate_
Data     1239206       7781  0.63%
Inst     3478202      17461  0.50%
Totl     4717408      25242  0.54%

active portion: 1.000000e+00
actv : 4743884  sleep:      0  nmreq:  2777779
cwait:  220987  iwait: 29478  total: 77721282
wbacc:   20966  wbful:   403

msize:   1024kB  EXCLK:     0Hz  bsize:      8-w
midle:  3601330  maddr:   45282  mdata:   239452
mwait:        0  mothr:       0  mtotl: 3886064
meff :  84.097%  mutil: 7.327%
```

Debug Info to Verify Correct Operation

System Summary

Instruction Profiling

Configuration Info

Activity Profiling

Block Energy Consumption

Bus Energy Consumption

FIGURE 4.4 : Simulator Report File for Crypto Benchmark.

All design optimizations were parameterized where possible (e.g. cache-line length, write-buffer size, etc.), so that previous configurations could be re-simulated via command line flags. This was crucial since the hardware capacitance models were

refined as the design progressed. Once the simulator incorporated the new capacitance models, all previous configurations could be resimulated to find any design optimization points that have shifted due to the newly refined models.

4.2.1 Capacitance Models

Energy consumption is one of the desired outputs from the simulator. To accommodate the variable V_{DD} in DVS, the underlying simulator models are based upon capacitance. During simulation, the capacitance is multiplied by V_{DD}^2, which varies depending upon the current clock frequency setting. There are varying approaches to capacitance estimation, with the simulator complexity increasing with estimation accuracy. The three simplest approaches are:

White-noise Approach (0th order): Every time a block or bus is accessed, a counter associated with that block is incremented by the average capacitance switched on that block/bus by V_{DD}^2. The switched capacitance values assume random signals on the blocks so that a single value can be used.

Data Correlation Approach (1st order): This approach uses simulator state variables associated with each block/bus's input ports to hold the previous state. The current and previous input values are XORed to count the number of transitioning bits, which are then multiplied by a capacitance per bit value. For a bus, this capacitance is just a sum of the estimated interconnect capacitance and fanout gate capacitance. For circuit blocks, an empirical estimate can be made from circuit simulation data. This works well on the majority of blocks that are bit-wise symmetrical, such as memory, register files, shifters, muxes, etc. Non-symmetrical structures that have inter-bit data correlation, such as an adder, will give an estimate with much greater variance because the capacitance is not only a function of how many bits transition, but the location of the transitioning bits as well.

Data/Instruction Correlation Approach (2nd Order): This approach builds

on the previous approach by also keeping track of previous instruction(s) with simulator state variables which can be used to model the inter-instruction dependencies. This better accounts for energy consumption in various muxes and state latches which are used to route the data flow of the processor from instruction to instruction.

Previous work has shown that the white-noise approach does an inaccurate job of energy estimation, so to provide a reasonable (first order) estimate, data correlation must be accounted for. The second order approach will improve accuracy, but is not necessary to provide useful energy estimates.

4.2.2 Implementation

It is critical that the energy estimation code be efficiently implemented so that the speed of the C simulator is not significantly degraded, since speed is the primary reason for using it. Thus, the counters and state registers required should utilize native integer word types on the host simulation machine. In addition, all functions should be compact and inlined to remove the unnecessary overhead of function calls.

A typical processor simulator consists of a large case statement representing the various instructions, or classes of instructions. As such, the energy estimation function calls required to annotate the simulator must be organized along instruction boundaries, and not physical organization. A second level of functions calls are used to map the logical system organization to an orthogonal microarchitecture specification,

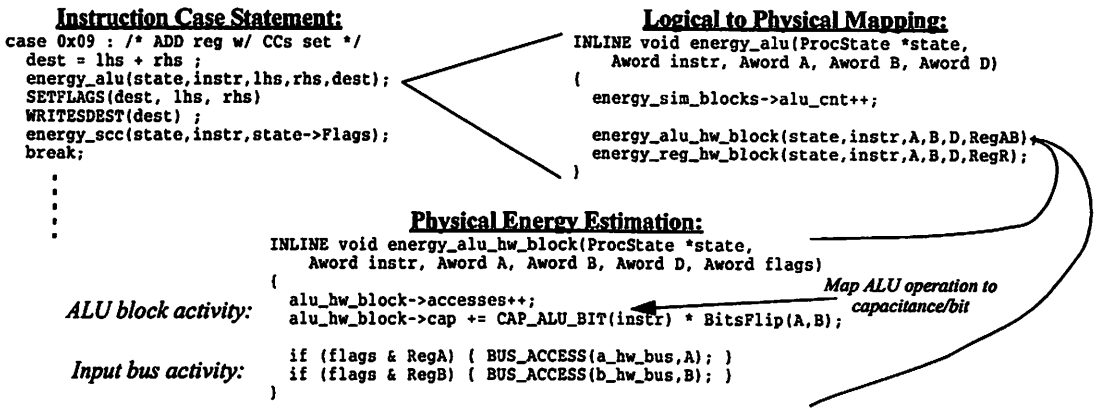


FIGURE 4.5 : Energy Estimation Implementation Example.

as demonstrated with the simple code example in Figure 4.5.

Fully annotating the simulator requires placing these logical energy function calls wherever the simulator changes the system state (e.g. *dest* = *lhs* + *rhs*). Using a simple system microarchitecture specification (Figure 4.6), these logical functions are then mapped to physical function calls.

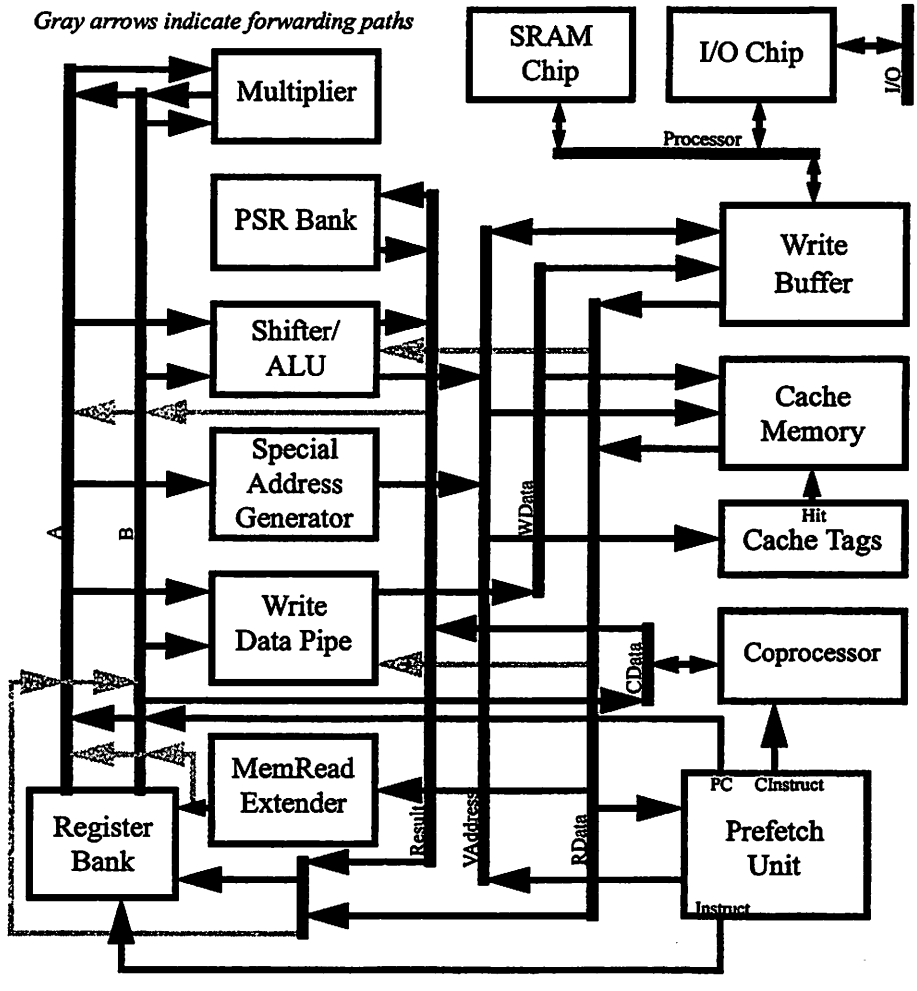


FIGURE 4.6 : Microarchitecture Model for High-level Energy Estimation.

4.3 Clocking Methodology

A well-defined clocking strategy is required for any sub-micron digital integrated circuit. The goal of this strategy is to limit the clock skew between any two latches. The primary reason for controlling the skew is to prevent race conditions

between latches, which lead to chip failure. The secondary reason is to prevent performance degradation. If the delay on a critical path is more than targeted, due to clock skew, then the overall cycle time needs to be increased for proper functionality.

Maintaining DVS compatibility adds further constraints to the clocking methodology. The ratio of clock skew to cycle time should remain fixed over the range of operating voltage. While the delay through the clock buffers will scale, the RC delay on the interconnect does not. Hence, the critical design corner to meet is at high voltage, fast process, and low temperature.

4.3.1 Latch Design

The behavioral model for the processor core (from ARM Ltd.) that was used as the design starting point assumed a latch-based design. Thus, the basic state element of the design was necessarily a latch, rather than a flip-flop.

While a static latch is not the most energy-efficient latch implementation, it is the most robust [west93][mark00]. It is well suited for use with aggressive clock gating (Section 6.1.3) because the clock can be held either high or low indefinitely without inducing a logical error. In contrast, a gated dynamic latch can have its internal state flipped via leakage currents and DVS voltage changes [burd94][mark00]. Additionally, static latches allow the system implementation to be stepped phase by phase, which is a tremendous advantage when debugging the hardware design. Thus, a fully static latch approach was taken.

Intuitively, it would seem that a single-phase latch is preferable over the traditional two-phase, cross-coupled latch because one less clock wire is needed. However, the natural design of a single-phase latch is dynamic. To make the latch static requires a much more complex latch design, and effectively eliminates its advantage.

The basic latch design is shown in Figure 4.7, in which the feedback device is

an inverter. Another possible implementation is to use a transmission gate between nodes *out* and *x*, but this can lead to a race condition on back-to-back latches if there is overlap on *clk* and *clkb*. The feedback device is clocked to both guarantee latch operation and speed up the latch. While the latch is transparent, there is no contention on the forward path, which greatly reduces the setup time, and allows any input signal to successfully change the value on node *x*. This does come at a penalty of an additional 15% in clock net capacitance for a datapath latch. The penalty will be less for a standard cell latch, where the interconnect capacitance is greater, thereby reducing the contribution from the feedback device.

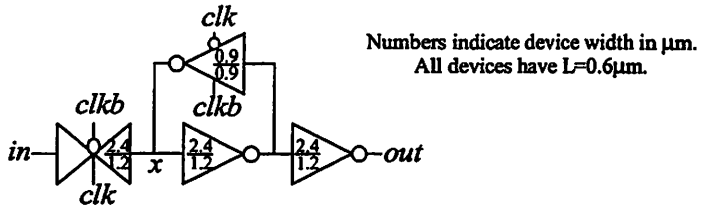


FIGURE 4.7 : Basic Two-phase Latch Design.

Since the behavioral model constrained the basic state element to be a latch, the fully-static approach seemed to best balance energy efficiency and design robustness. Had the model assumed a flip-flop based design, then a pseudo-static version [burd94] of the TPSC register [yuan89] would yield a more energy-efficient and robust design. This solution requires only one clock wire, yields equivalent register setup & hold times, and allows clock gating if the clock is held low.

4.3.2 Clock Architecture

The basic clock-gating strategy is split-level. There is a local enable signal for each n-bit latch, or set of latches. This signal is active-high, and is used to selectively generate a local clock pulse. There are also three global halt signals, which are used to halt large sections of the microprocessor: the processor core, the cache subsystem, and the bus interface. Rather than actively suppressing clock pulses, which will change the processor state, it simply gates out the low-phase of the global clock signal, as

demonstrated in Figure 4.8, maintaining exact processor state.

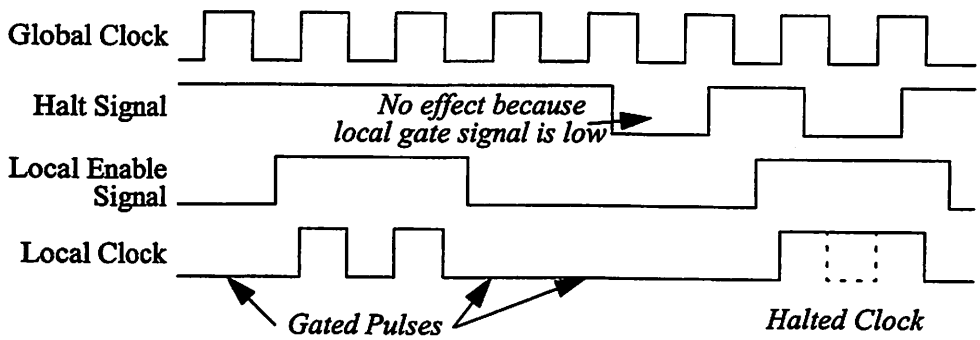


FIGURE 4.8 : Clock Gating & Halting Timing.

This halting mechanism allows the processor memory interface to be designed to expect the cache subsystem to return a word within one cycle of the request. If the cache subsystem needs to go out to main memory, it simply halts the processor core until it has the desired word available, and eliminates any unnecessary switching activity in the core. Likewise the bus interface can stall the cache subsystem when it is retrieving words, and the external I/O system can stall the bus interface when a high-latency I/O memory request is made.

The natural implementation for this would be a three-level clock hierarchy, divided by the global (halt) clock drivers, and the local (enable) clock drivers. However, the skew can be better controlled in a two-level clock hierarchy by removing the skew contribution from the global clock driver, although this comes at the expense of increased energy consumption.

In the two-level hierarchy, the total capacitance on the global top-level clock net is 10pF, which is just 3% of the nominal effective capacitance switched per cycle. Another consideration is clock power when the processor is in the sleep mode. However, if the processor is put into a low-speed mode before entering sleep, the idle clock power dissipation is only 72μW, which is only 9% of the total system sleep mode power. Hence, it was decided a worthwhile trade-off to implement a two-level clock hierarchy which provided a more controlled clock skew. In addition, it made the timing

analysis simpler, because both the local enable and global halt signals are terminated at the same gate.

4. 3. 2. 1 Clock Drivers

One approach to generating the two-phase clock signals is to locally invert within the latch cell. However, this has two significant drawbacks. First, the additional gate gets toggled every clock cycle, which adds more capacitance to the global clock net as compared to the interconnect capacitance added by running a second clock wire. Second, the skew between the two clock signals is now a full gate delay, which is not tolerable as will be demonstrated in the clock skew analysis later on. To reduce this skew, the non-inverted clock signal can be buffered by two inverters, but since these toggle every cycle, further capacitance is added to the global clock net. Hence, it is preferable to generate the two-phase signal once per n-bit latch. For the datapath registers, which dominate the overall register count, this requires one clock driver per 32-bit latch, and can be designed to have well-controlled skew.

Before each n-bit latch is a clock driver which performs single to differential polarity conversion, as shown in Figure 4.9. A phase-1 clock driver generates a high pulse on the *Phi1* signal while *clock* is low, and the *local-enable* and *global-halt* (active

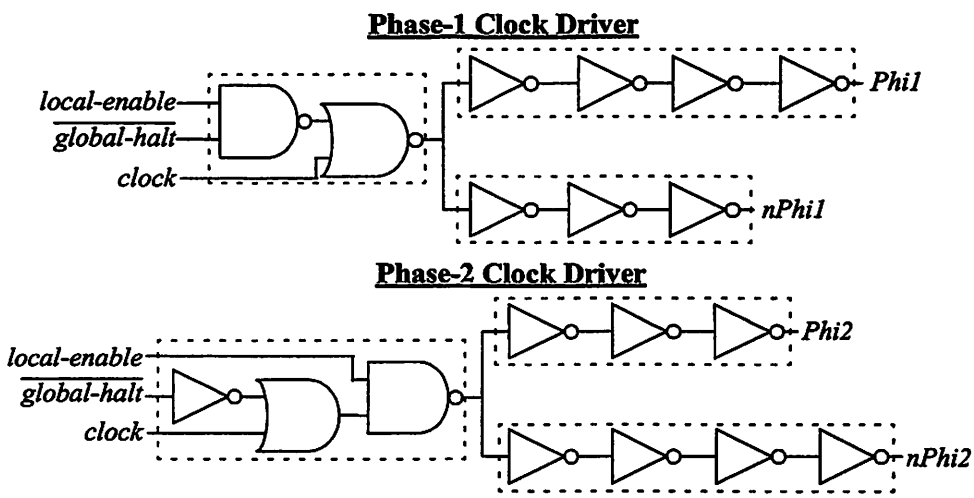


FIGURE 4.9 : Modular Clock Driver Design.

low) signals are high. Similarly, the phase-2 clock driver generates a high pulse on *Phi2* when *clock* is high, and the *local-enable* and *global-halt* signals are high. The *local-enable* signal must only transition in the phase opposite of the clock driver phase (e.g. a phase-1 clock driver's *local-enable* signal can only change while *clock* is high, such that it must be generated by a phase-2 latch, or derived only from signals output by phase-2 latches). The *global-halt* signal, which drives clock drivers of both polarities, can only transition while *clock* is high.

The maximum skew between any two clock drivers was determined in Section 4.3.4 to be less than one-half of a gate delay. To maintain controlled skew, the inverter chains need to be carefully sized for the output load, which can vary from less than 50fF to 1pF. Close to 150 clock drivers where used in the design. Rather than design a new clock driver each time one was required, and insure that it met the skew constraint over process and voltage, a modular clock driver design was developed.

This modular approach consists of 2 gate cells, one for each phase, and 24 each of inverting, and non-inverting driver chains. The modular cells in Figure 4.9 are denoted by dotted lines. The input capacitance to the driver chain was kept constant so that the same gate cell could be used for both small and large drivers, and not require re-tuning. Since the input inverter is a fixed size, and the output inverter size is solely dictated by the load being driven, the non-inverting clock path required four inverters, rather than just two, to properly tune the driver delay. However, the extra two inverters consume an insignificant amount of energy on all but the smallest clock drivers. In addition, the layout took a modular approach, with the gate cell in the middle and a driver chain cell abutting either side of it. The most significant benefit of this modular approach was that the 50 unique cells could be designed and verified independent of the final chip design.

Table 4.1 summarizes the delay and skew of the entire clock driver suite at three different voltages and across process variation. The values have been normalized

to the clock cycle time, t_{CLK}, to demonstrate that both the mean delay, and the skew track extremely well over V_{DD}.

TABLE 4.1 Clock Driver Delay and Skew (0.6μm Process).

V_{DD}	Process Corner	Delay (ps / % of t_{CLK})			Skew (ps / % of t_{CLK})
		Mean	Max.	Min.	
4.1	Fast	555 / 7.9%	590 / 8.4%	510 / 7.3%	80 / 1.1%
	Nominal	709 / 7.9%	755 / 8.4%	650 / 7.2%	105 / 1.2%
	Slow	897 / 7.8%	960 / 8.3%	826 / 7.2%	134 / 1.2%
3.3	Fast	648 / 8.6%	687 / 9.2%	601 / 8.0%	86 / 1.1%
	Nominal	835 / 8.4%	885 / 8.9%	770 / 7.7%	115 / 1.2%
	Slow	1070 / 8.6%	1140 / 9.1%	989 / 7.9%	151 / 1.2%
1.1	Fast	5510 / 7.3%	5970 / 8.0%	5100 / 6.8%	870 / 1.2%
	Nominal	8870 / 7.4%	9640 / 8.0%	8080 / 6.7%	1560 / 1.3%
	Slow	17300 / 4.5%	19300 / 5.1%	15000 / 3.9%	4300 / 1.1%

When a new clock driver was designed into the chip implementation, the capacitance on both *Phi* and *nPhi* were estimated from the schematic. These values were used to index into Table 4.2 to determine the driver size for the two signals. If the desired clock driver cell currently existed, it would be used, and if not, then a new clock driver can be rapidly constructed. Once the layout was complete, the design was extracted to measure the final signal capacitance. If the sizings were found to be wrong for the extracted capacitance, a new clock driver could easily be swapped in.

TABLE 4.2 Clock Driver Sizing (not including diffusion capacitance)

Driver	Min Load (fF)	Max Load (fF)
1x	37.5	62.5
2x	50	87.5
3x	75	125
4x	112.5	187.5
5x	150	225
6x	187.5	262.5
7x	225	300
8x	262.5	337.5
9x	300	375
10x	337.5	412.5
11x	375	450
12x	412.5	487.5

Driver	Min Load (fF)	Max Load (fF)
13x	450	525
14x	487.5	562.5
15x	525	600
16x	562.5	637.5
17x	600	675
18x	637.5	712.5
19x	675	750
20x	712.5	787.5
21x	750	825
22x	787.5	862.5
23x	825	900
24x	862.5	937.5

It would seem that only 48 unique clock drivers are needed, assuming the output signals having equal driver strength (1-24x for phase-1, and 1-24x for phase-2). However, the majority of clock drivers had differently sized driver chains due to varying interconnect capacitance. Hence, this modular design approach worked quite well by allowing the rapid assembly of any arbitrarily-sized clock driver.

4.3.3 Bounds on Allowable Skew

The worst-case race condition between back-to-back latches sets the maximum allowable clock skew [west93]. Any logic in between the latches makes the circuit more robust against clock skew. There are two different cases to consider. The same-latch case occurs when the two latches are clocked by the same two clock signals, such as in a flip-flop, and there is skew between *Phi* and *nPhi*. The unrelated-latch case occurs when there are two sets of clock signals, *PhiA/nPhiA* and *PhiB/nPhiB*, and the skew occurs between the two sets. For each of these cases, the maximum allowable clock skew, t_{MAX}, was found so that the two latches remain immune to clock race, as shown in Figure 4.10.

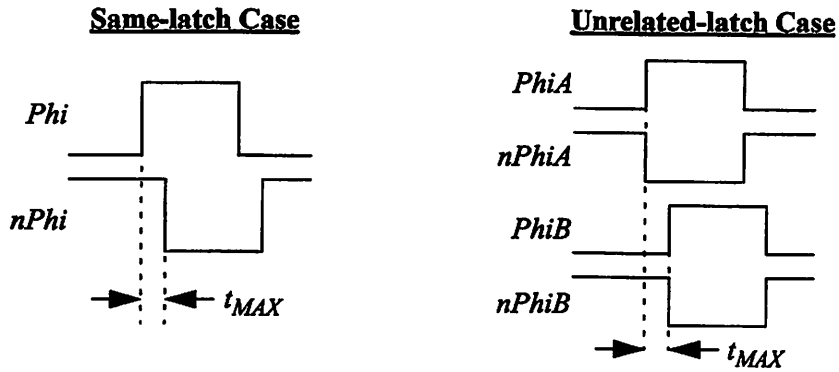


FIGURE 4.10 : Maximum Allowable Clock Skew Measurement for Race Immunity.

Spice simulations were run to measure t_{MAX} over voltage and process, as given in Table 4.3. As previously mentioned, the critical corner to design for is high voltage and fast process because interconnect RC delay will be most significant there. The other corners were measured to ensure that the clock drivers scaled properly over voltage.

These results are used in Section 4.3.5 to verify that there will be no fatal errors in the chip implementation due to race conditions.

TABLE 4.3 Maximum Allowable Clock Skew (in ps).

V_{DD}	Process Corner	t_{MAX} (unrelated-latch)	t_{MAX} (same-latch)
4.1	Fast	320	300
	Nominal	410	390
	Slow	530	490
3.3	Fast	380	380
	Nominal	500	480
	Slow	640	610
1.1	Fast	3700	3800
	Nominal	6000	6100
	Slow	10500	15000

4.3.4 Sources of Skew

There are several sources of clock skew, and each component was carefully analyzed and quantified. A total maximum clock skew can be calculated by summing up the individual components. While this is a conservative estimate because it assumes no correlation in skew, it has a high confidence level because each component has been completely analyzed and accounted for. The location of these sources are shown in Figure 4.11.

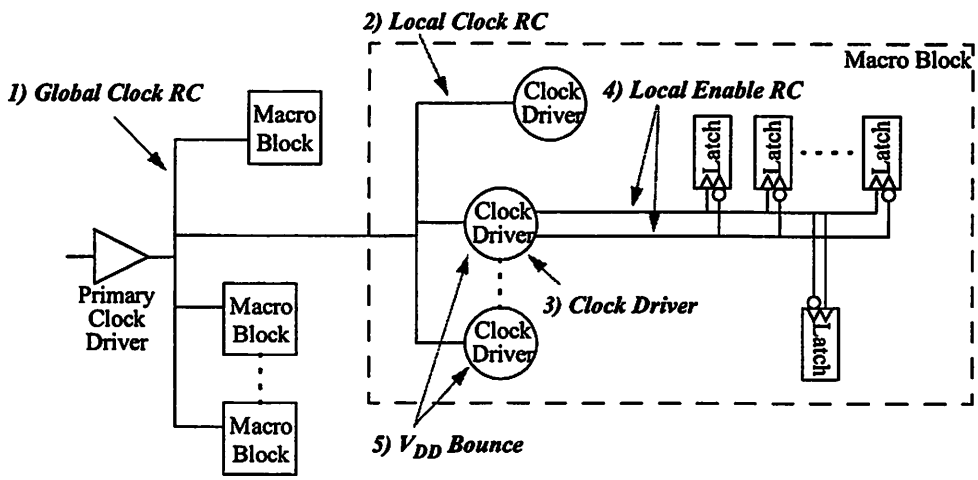


FIGURE 4.11 : Five Sources of Clock Skew.

The initial analysis assumed that the two dominant sources of skew were the global clock RC and the clock driver skew. Each component was given a delay budget

equal to 25% of the maximum allowable skew. This gave 100% headroom for margin of error and for additional skew components. Upon completion of the design, the various components were re-analyzed to verify the non-existence of race conditions.

4. 3. 4. 1 Global Clock Wiring

This component is due to the voltage-independent RC delay on the global clock wire. It is measured as the skew between the clock input signals each macro block sees, where each macro block was modeled as a lumped capacitance. To make sure the delay budget of 25% of the maximum allowable skew could be met, an initial chip floorplan was analyzed. A very simplistic model demonstrated that this target could be met, and only require 3-4x larger than minimum-size wiring.

At the top level, widening the clock wire was very beneficial because the fractional capacitance of the clock interconnect was small compared to all the clock drivers' gate capacitance. Hence, widening the wire almost linearly reduced the RC delay.

During the course of the implementation, a regular 20μm-wide clock routing channel was created to allow for post-layout wire widening, and to eliminate interline capacitance. The clock distribution attempted to model an H-tree distribution network. After the initial chip layout was completed, a parameterized model of the global clock wire was created that modeled the clock wire as a distributed RC network with lumped capacitances for each macro block. The widths of the various wire segments were parameterized so that the RC delay could be tuned without having to extract the layout for each iteration.

The tuned top-level clock routing for the prototype processor is shown in Figure 4.12. The routing used third-level metal (*Metal3*) almost exclusively, due to its low resistivity, except where the clock routing intersected the top-level power grid network. The final simulation yielded a maximum RC delay between any two macro

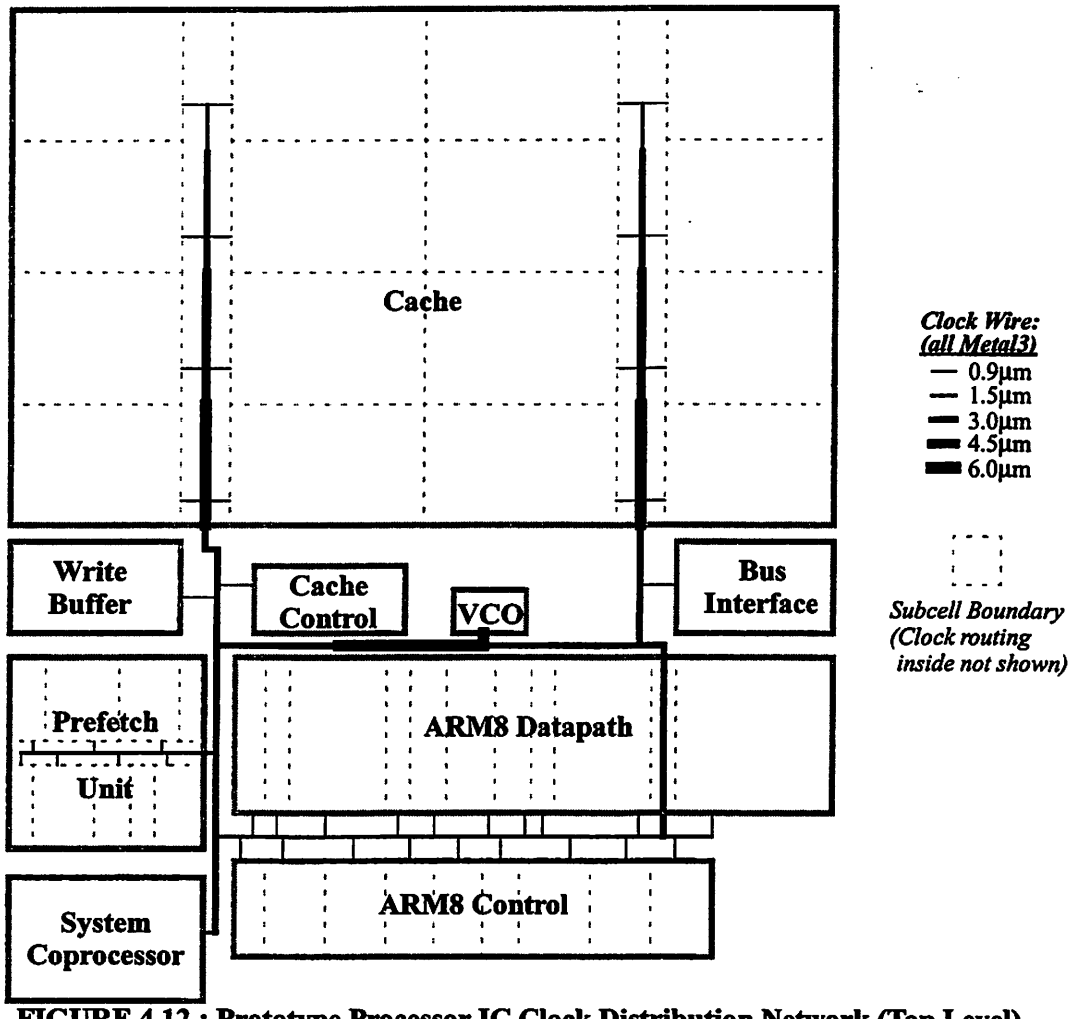


FIGURE 4.12 : Prototype Processor IC Clock Distribution Network (Top Level).

blocks of 81ps. The maximum delay between any two macro blocks within the processor core was just 31ps. The widest wire segment was 6.0µm, with an average wire segment of approximately 2.4µm.

4. 3. 4. 2 Local Clock Wiring

This component is the RC delay on the clock signal within each macro block, measured as the largest skew seen by any two clock drivers. This component was initially assumed to be small, and the first large control block was used to verify that. The ALU control block was simulated to have 15ps skew. To provide a margin for error, this skew component was set to 20ps.

This RC delay is negligible because the longest wire is no more than 1mm, as

dictated by the bounded macro block size. Care was taken in routing this clock signal; *Metal3* was predominantly used, with the minimum number of vias. The 15ps estimate above was for an initial place and route of the ALU control block with no special attention paid to the signal routing, increasing the conservatism of the estimate.

4. 3. 4. 3 Clock Driver Skew

The drivers are a dominant source of clock skew, and were designed to have a maximum skew across all possible drivers no more than 25% of the maximum allowable skew. Achieving this goal required careful design and sizing of the clock drivers. The largest driver was implemented first to ensure it could be designed, and then smaller drivers were designed to be within the delay budget. For the smallest drivers, additional internal capacitance had to be added to meet the targeted delay variation.

The 25% rule was most critical at high voltage, where the RC delay is also significant. At the low voltage corner, the allowable margin was increased to 50% since the RC delay becomes negligible. The maximum delay variations for the entire suite of clock drivers is given in Section 4.3.5, and meets the targeted specification.

4. 3. 4. 4 Local Enable Wiring

This component is the RC delay on the *Phi* and *nPhi* latch-enable signals. The worst case for this arises in the datapath, in which the enable signal must traverse 950μm across a 32-bit latch. This was measured to be 20ps, and relatively negligible. To provide a margin for error, this component was set to 30ps.

In the place and routed control blocks, care was taken in routing the enable signals. The placement was optimized to group latches around similar enable signals. Additionally, these routes were given the highest priority, along with the local clock signal, to optimize their routing. In the final layout of the synthesized blocks, due to the latch clustering, no enable signal ran longer than 1mm validating the 30ps estimate.

4. 3. 4. 5 Enable Rise/Fall Time

Because the enable signals have a finite rise/fall time, there is a finite time that the latches can remain open even if the signals' skew, as measured between 50% points, is zero. Shown in Figure 4.13 are Spice simulations results which demonstrate that the maximum allowable skew actually increases monotonically with rise/fall time, despite the increasing overlap of the enable signals. Hence, the maximum allowable skew, as measured with step edges in Section 4.3.3, is a conservative estimate of the skew.

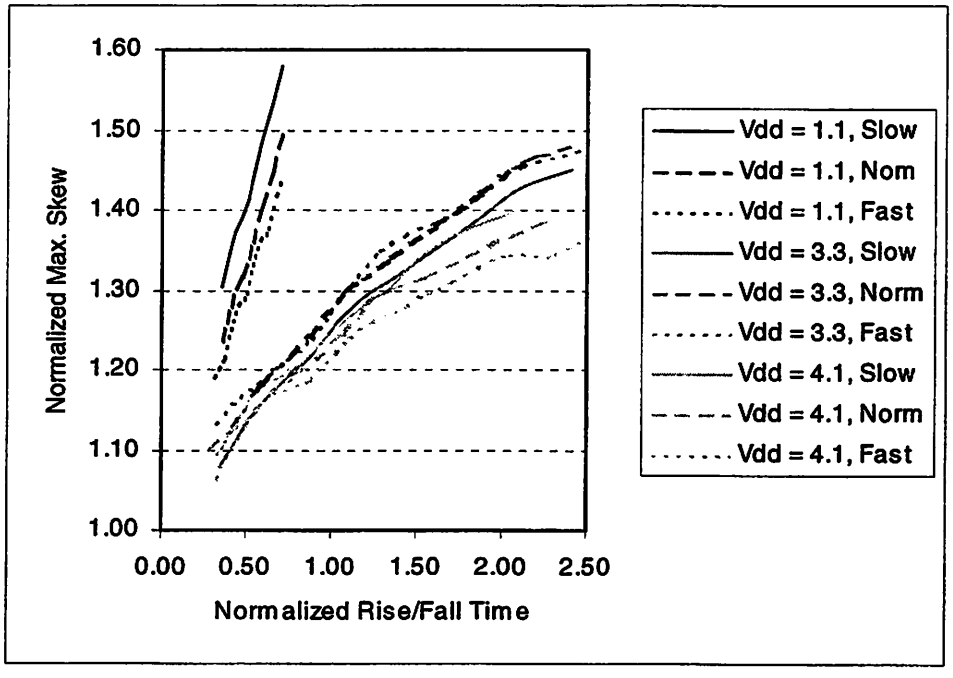


FIGURE 4.13 : Skew as a Function of Enable Rise/Fall Time.

This component, if anything, would contribute a negative number to the maximum skew calculation. However, it is simply ignored, increasing the conservatism of the maximum allowable skew calculation.

4. 3. 4. 6 V_{DD} Variation Skew

The power distribution network (Section 4.4) was designed so that no more than a 10% critical-path delay variation occurs with V_{DD} variation. V_{DD} variations only affect the clock driver delay, which has a mean delay just under 10% of the target critical-path delay. Thus, the maximum skew from V_{DD} variation is just a product of

these two percentages, or just 1%. This component is voltage dependent, and will vary with V_{DD} similar to clock driver skew.

4.3.5 No-race Verification

To verify that race conditions cannot exist, the individual skew components were summed up and compared against the maximum allowable skew. Due to the initial delay budgeting on the clock drivers, and the global clock wire analysis, the final implementation met the targeted skew with some room to spare.

Table 4.4 gives the comparison for the unrelated-latch case. As expected, the smallest skew headroom (Maximum Allowable Skew - Total Skew) occurs at high voltage and fast process, and is only 13% of the allowable skew. While this gave little margin for error, it seemed reasonable due to the enormous conservatism built into the estimate.

TABLE 4.4 Total Clock Skew Summary (all times are in ps): Unrelated Latches

Vdd	Process Corner	Global Clock RC	Local Clock RC	Clock Driver	Local Enable RC	V_{DD} Bounce	Total Skew	Maximum Allowable Skew
4.1	Fast	80	20	80	30	70	280	320
	Nom			105		90	325	410
	Slow			134		115	380	530
3.3	Fast			86		75	290	380
	Nom			115		100	345	500
	Slow			150		125	405	640
1.1	Fast			870		750	1750	3700
	Nom			1560		1200	2890	6000
	Slow			4300		3800	8230	10500

The same-latch case was much easier to meet, as shown in Table 4.5. Since in this case the two latches share the same *Phi* and *nPhi* signals, global and local clock RC are made irrelevant because the latches share the same clock driver.

Thus, by design, the chip implementation should be free from any race conditions, across the entire voltage and process range. The only precondition is that

TABLE 4.5 Total Clock Skew Summary (all times are in ps): Same Latch

Vdd	Proc.	Clock Driver	Local Enable RC	V_{DD} Bounce	Total Skew	Max Allowable Skew
4.1	Fast	80	30	70	180	300
	Nom	105		90	225	390
	Slow	134		115	280	490
3.3	Fast	86		75	190	380
	Nom	115		100	245	480
	Slow	150		125	335	610
1.1	Fast	870		750	1650	3800
	Nom	1560		1200	2790	6100
	Slow	4300		3800	8130	15000

the latch contains two inverters. The number of inverters were reduced in some latches due to critical path constraints, but it had to be ensured that there was no latch immediately following a sped-up latch.

4.4 Power Distribution Methodology

Typical low-power chip designs have very relaxed constraints on the power distribution network due to low DC and peak supply currents. While a DVS processor generally has a low DC supply current, the peak supply current can be quite high when it is operating at maximum clock frequency and supply voltage, thereby placing tight constraints on the power distribution network, both at the chip level and at the board level. Thus, power distribution requires careful design consideration in a DVS system similar to any high performance, and high current, chip design.

4.4.1 On-chip Supply Variation

The on-chip voltage supply will vary due to inductive and/or resistive voltage drops across the chip's power distribution network, and across the pins and bonding wires of the chip's packaging. Global supply variations, which occur uniformly across the chip, are essentially no different than changing the external supply voltage in a DVS system. Thus, the DVS chips are relatively immune to global on-chip supply variations.

However, the problems arising from local supply variations, which occur within a limited area of the chip, are the reduction of signal noise margin and timing violations, both of which can induce functional failure.

4. 4. 1. 1 Noise Margin Reduction

Static CMOS circuits and most dynamic logic circuits (e.g. Domino, NORA, DCVSL, etc.) are very robust against noise margin reduction, since their signal swing is the full value of V_{DD}, and have a noise margin of at least V_T. Memory arrays (e.g. RAM, ROM, PLA, etc.) are more susceptible to noise margin reductions. To make them more robust, they should be designed to be either differential, or full swing. In the prototype system (Chapter 7), the only types of memory arrays used were RAMs and CAMs, which were designed with differential bitlines for improved robustness. The critical circuits which are most susceptible to noise margin reduction are the I/O transceivers due to their very large, and localized, peak currents.

4. 4. 1. 2 Timing Violations

As described in Section 3.3.4, local on-chip supply variations can lead to timing violations if a critical path sufficiently slows down to exceed the clock cycle time of the ring oscillator. A DVS processor cannot have timing violations induced by global supply variations, since the ring oscillator's delay (e.g. the inverse of the clock frequency) will vary with the delay of the critical paths.

A design margin of 5% was included in the timing verification of the processor to account for localized voltage drops. The equation for delay sensitivity (Equation 3.9), which is the relative change in delay(V_{DD}) for a given ΔV_{DD}, can be rewritten to translate this 5% design margin into a maximum allowable voltage drop, ΔV_{DD}, at a given value of V_{DD}:

$$\Delta V_{DD}(V_{DD}) \approx \frac{\frac{\partial Delay}{Delay}(V_{DD}) \cdot Delay(V_{DD})}{\partial Delay / \partial V_{DD}} = \frac{5\% \cdot Delay(V_{DD})}{\partial Delay / \partial V_{DD}} \qquad \text{(EQ 4.1)}$$

and this can be used to calculate the maximum resistance, R_{MAX}, allowed on the supply line given a gate's peak output current, I_{GATE}:

$$R_{MAX} \approx \frac{5\% \cdot Delay(V_{DD})}{\partial Delay / \partial V_{DD}} \cdot \frac{1}{I_{GATE}(V_{DD})} \quad \text{(EQ 4.2)}$$

If the supply distribution network is designed so that the resistance that the gate sees to a solid, reference voltage is less than R_{MAX}, then any delay variation due to supply variation will fall within the 5% design margin, and the processor will continue to operate correctly without any spurious timing violations.

4.4.2 Chip-level Distribution

The chip-level power distribution network for the processor chip is shown in Figure 4.14. The other DVS-compatible chips in the processor system (SRAM and I/O chips) were designed in similar fashion, but the processor chip is focused on in detail to demonstrate the power distribution methodology. There are two separate power supplies, V_{DD} for the core, and V_{DDIO} for the pad ring of the chip, to isolate the core circuitry from the I/O transceivers. There is a single ground on the chip, since the low-impedance substrate of our process makes separate ground supplies difficult to isolate.

There were two primary design goals for the power distribution network. First, there should be sufficient bypass capacitance on the chip to supply the charge for the large switching currents. This will reduce the voltage drop across the bonding wires and packaging pins, which would otherwise be intolerable. This is not an issue for traditional low-power/low-voltage chips, but it is an issue for a DVS-compatible chip which can have large switching currents at high voltage. The second goal was to ensure that any point on the chip has a low-impedance connection to a solid voltage reference, either a large on-chip bypass capacitor or the external voltage supply. This was critical to eliminate timing violations due to localized voltage drops.

Ground is routed in *Metal3* directly over power in *Metal2*, except where the

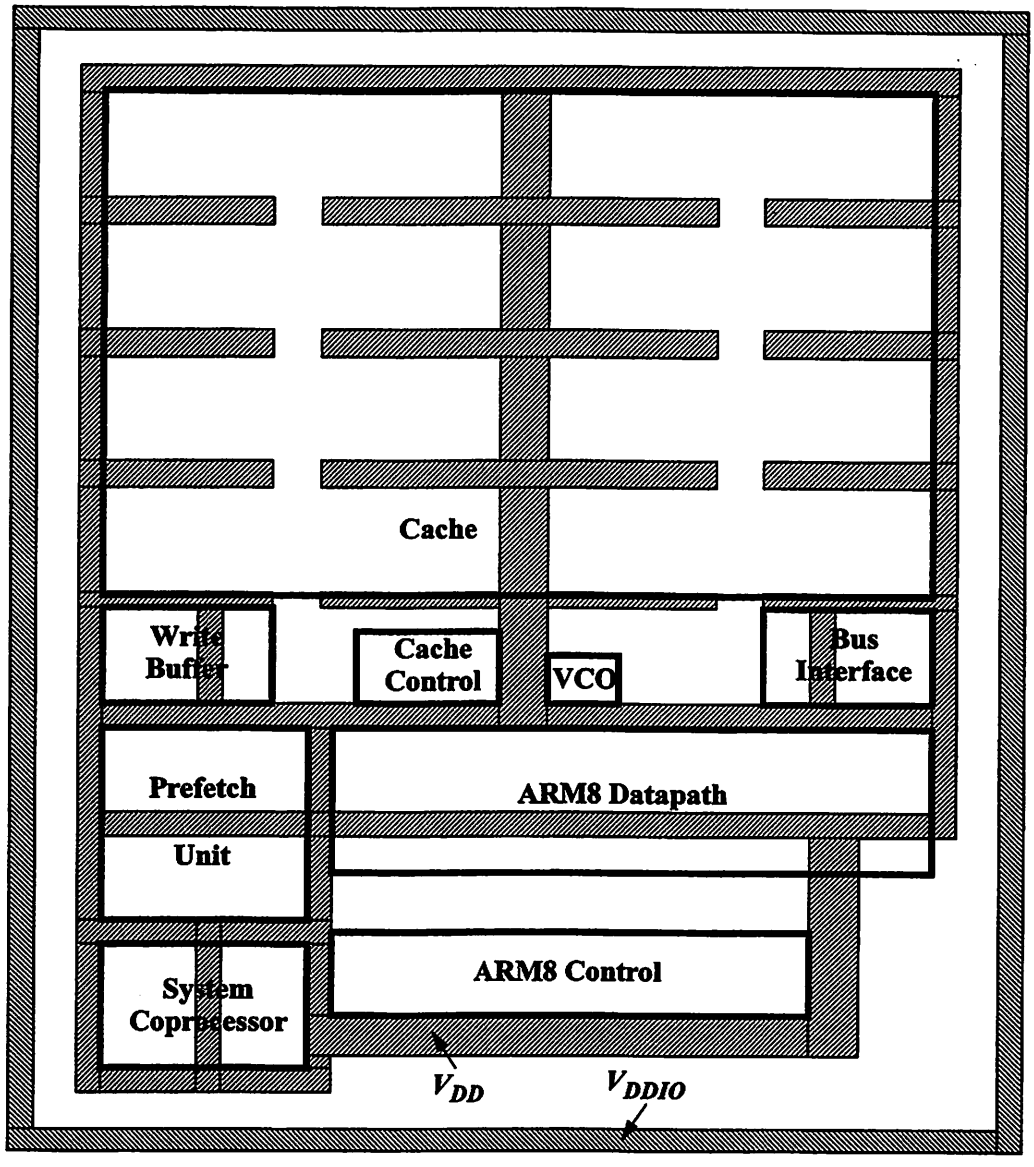


FIGURE 4.14 : Prototype Processor IC Power Distribution Network.

power lines dissect a block, in which they are routed in *Metal3* and *Metal1*, leaving *Metal2* to be used for signal routes. Routing ground on top of power helps to both maximize the inter-metal bypass capacitance, and minimize inductive losses via magnetic field cancellation.

A total of 16 pins for V_{DD} and another 10 pins for V_{DDIO} are distributed uniformly around the periphery of the chip. The large number is essential to reduce the inductive voltage drop on the bonding wires, as described in more detail in Section 7. 2. 8. 2. Additionally, by spreading them evenly around the chip's edge, any

point on the V_{DD} and V_{DDIO} networks sees minimal resistance to the off-chip power supply.

4. 4. 2. 1 Bypass Capacitance

The bulk of the bypass capacitance is provided by NMOS devices which provide the highest capacitance per area. These devices were placed in all the significantly-sized open areas of the chip as well as under the power lines themselves. Power is connected to the gate, while the source and drain are tied to ground, maintaining the device in the triode region of operation.

The width and length of the NMOS devices are constrained in size due to resistance in the device channel and in the polysilicon gate. The maximum RC time-constant of the channel is to its mid-point, and is:

$$\tau_{CHAN} = \frac{1}{4} \cdot R_{DS} \cdot C_{GS} = \frac{1}{4} \cdot \frac{(W \cdot L \cdot C_{OX})}{k_p \cdot (W/L) \cdot (V_{GS} - V_T)} = \frac{L^2 \cdot C_{OX}}{4 \cdot k_p \cdot (V_{DD} - V_T)} \qquad \text{(EQ 4.3)}$$

where the factor of four occurs because the source and drain are connected, dividing the effective R and C by a factor of two each. The time-constant scales inversely with V_{DD} similar to circuit delay, so that the maximum channel length is not strictly limited at maximum supply voltage and current draw, but relatively optimal over all V_{DD}. The maximum time-constant of the polysilicon gate is:

$$\tau_{GATE} = \frac{1}{4} \cdot R_{GATE} \cdot C_{GS} = \frac{1}{4} \cdot \left(\rho_{PLY} \cdot \frac{W}{L}\right) \cdot (W \cdot L \cdot C_{OX}) = \frac{\rho_{PLY} \cdot W^2 \cdot C_{OX}}{4} \qquad \text{(EQ 4.4)}$$

where ρ_{PLY} is the sheet resistance of polysilicon (10 Ω/sq. in our 0.6μm process) and the factor of four occurs because the gate is contacted on both sides of the device. Unlike τ_{CHAN}, τ_{GATE} is independent of V_{DD} so the maximum channel width is strictly limited at maximum voltage.

For the core circuitry (powered by V_{DD}), the current spikes are on the order of 50-100ps wide at maximum voltage. To make the RC delay of the bypass capacitance

negligible, the maximum time-constant value was set to 25ps. This dictated the maximum *W/L* of a bypass NMOS device to be 54/3 μm in our 0.6μm process. When including the overhead for routing the bypass capacitors, the area efficiency of this size bypass device is approximately 62%.

Since all the power metal lines shown in Figure 4.14 contain bypass capacitors underneath them, and the network is connected with cross-bar metal lines between blocks, the entire V_{DD} network provides a solid voltage reference. That is, the voltage on the V_{DD} network varies approximately uniformly across the chip.

Providing a solid voltage reference for V_{DDIO} is a much more difficult task. For the worst-case condition that all I/O pins transition low to high, and they drive the maximum 50pF external load capacitance, the total capacitance charged in a cycle can be as high as 2nF. Bypass capacitance is used to mitigate the voltage drop on V_{DDIO}, placed both under the V_{DDIO} power lines and in all available open areas between pads. But, only 2nF of capacitance could be placed on-chip, such that for the worst-case condition, the on-chip bypass capacitance cannot supply all of the charge, leading to localized voltage drops on the I/O transceivers.

However, the bus interface was designed to have the minimum amount of gates in its path delay to provide the requisite design margin for very large increases in delay driving the external bus. By eliminating the bus interface as a critical path, large voltage drops on V_{DDIO} are tolerable without inducing timing violations. Thus, only the core circuitry connecting to the V_{DD} network had to be carefully designed to guarantee no timing violations by design.

4. 4. 2. 2 Local Supply Routing

As was demonstrated in Section 3.3.4, the delay sensitivity to V_{DD} is a maximum at $2 \cdot V_T$ (approximately 2V in our 0.6μm process), so the local supply network must be designed at this value of V_{DD}, which will determine the smallest amount of

resistance tolerable in the power supply network. At 2V, Equation 4.2 is roughly 0.05/ I_{GATE}. This equation can be expressed in terms of NMOS device width (W_N) and number of gates in parallel (N_{GATE}) for 0.6µm process as:

$$R_{MAX} \approx \frac{0.05}{N_{GATE} \cdot v_{SAT} \cdot C_{OX} \cdot W \cdot (V_{DD} - V_T - V_{Dsat})} = \frac{0.001}{N_{GATE} \cdot W_N} \qquad \text{(EQ 4.5)}$$

The equivalent resistance for a PMOS device is for a device with a width is 2.5 times as large as the NMOS device due to the decreased mobility.

This equation was used to determine how wide the local power routes to the core circuitry should be. For the 32-bit datapaths of the ARM8 core, the prefetch unit, and the coprocessor, the 100µm-wide *Metal2* and *Metal3* lines of the power distribution network are spaced 825µm apart as shown in Figure 4.15. Within the datapath, *Metal2* and *Metal3* are used for signal routing such that power had to be routed to the individual bits in parallel *Metal1* lines. The maximum resistance to the power distribution network is from the midpoint of the datapath, whose resistive losses can be modeled as shown.

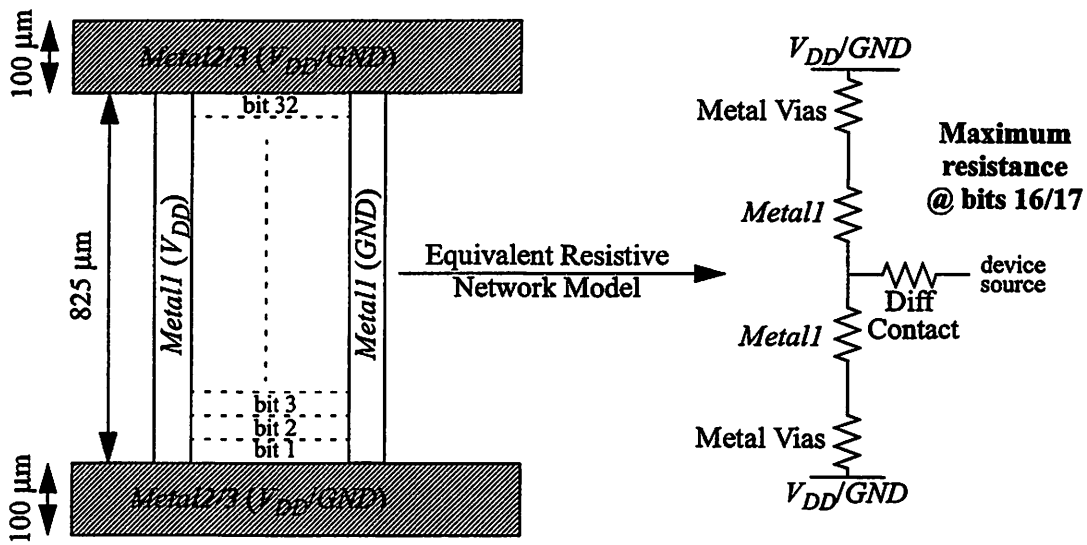


FIGURE 4.15 : Local Datapath Power Routing.

For a simple datapath cell with minimal output load (a 1x gate), W_N = 1.2µm. A worst-case analysis must assume all 32 bits of the datapath switch at once, which

yields $R_{MAX} = 26\Omega$. The contacts, vias, and metal wires must be designed so that:

$$R_{MAX} > R_{DIFF} + \frac{1}{2}(R_{MET1} + R_{VIAS}) \qquad \text{(EQ 4.6)}$$

For our process technology, R_{DIFF} will always be approximately 20% of R_{MAX}, because the number of parallel diffusion contacts scales with W_N, so that R_{DIFF} scales down with R_{MAX}. The metal via resistance, R_{VIAS}, can be made negligible (<2% of R_{MAX}) because many vias can be added in parallel under the 100µm wide power network. Thus, the width of *Metal1* must be adjusted so that:

$$R_{MET1} < 1.6 \cdot R_{MAX} = \frac{50\mu}{W_N} \qquad R_{MET1} = \frac{L_{MET1}}{W_{MET1}} \cdot \rho_{MET1} = \frac{31\mu}{W_{MET1}} \qquad \text{(EQ 4.7)}$$

and combining these two relations for R_{MET1} gives:

$$W_{MET1} > 0.6 \cdot W_N \qquad \text{(EQ 4.8)}$$

which is a concise rule-of-thumb for sizing the *Metal1* power lines given the size of a gate's transistors. By following this, the datapath could be implemented to not have any potential timing violations, due to power line resistance, by design.

Similar equations were developed for the large cache memory arrays, as well as the placed-and-routed standard cells of the core control logic. For the standard cell arrays, the cell-level power was routed in *Metal1*, as shown in Figure 4.16, again to free

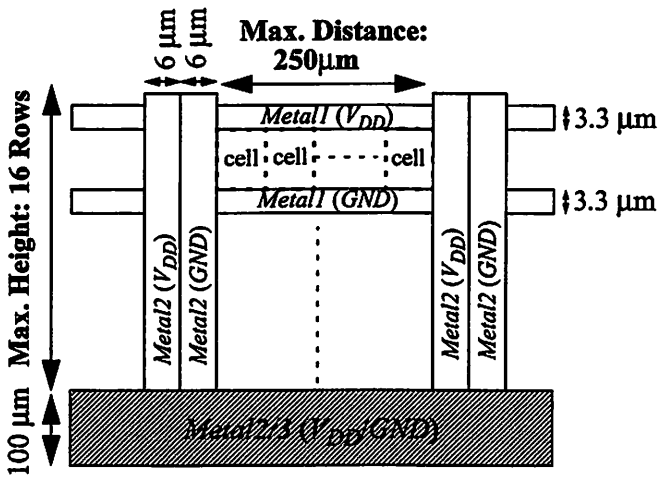


FIGURE 4.16 : Local Standard Cell Power Routing.

up *Metal2* and *Metal3* for signal routing. However, to reduce resistance to the power distribution network, as necessary, 6μm-wide *Metal2* straps connected up the power lines, and could be spaced no further than 250μm apart. The maximum number of rows was calculated to be sixteen. Again, by following these constraints, any place-and-routed standard cell implementation could be guaranteed to not have potential timing violations by design.

4.4.3 Board-level Distribution

The DVS regulator chip [stra98] is very sensitive to board-level parasitics which may interact with the output LC filter for the DVS supply voltage, V_{DD}. Thus, careful attention had to be paid in laying out the power distribution for the board. An entire PCB plane was dedicated to V_{DD}, which reduced the parasitic resistance to a negligible amount. This also allowed a single PCB via to connect the V_{DD} pins on the chip sockets to the power plane, in order to minimize the inductance.

Since the plane provides negligible resistance and inductance, the need to place the filter capacitor next to the inductor was eliminated. Instead, the 5μF capacitor was evenly spread out across the board, and placed next to the chips' V_{DD} pins as 100nF and 200nF bypass capacitors. This provided both the necessary filter capacitance, as well as good bypass capacitance, to eliminate unnecessary noise on the chips' V_{DD} pins.

4.5 Functional Verification

At every stage in the design process, the design specification was constantly verified for functional correctness. At the higher, more abstract, design levels, logical behavior is checked. Closer to the physical design, individual signals are checked for phase correctness and setup/hold times in additional to logical behavior. A verification methodology was developed so that test code had to be developed only at the top design level. At all subsequent design levels, scripts automatically generated new test vectors

from the previous design level's test vectors.

4.5.1 Behavioral Verification

The essence of behavioral verification is ensuring that the processor system properly operates from the programmer's point-of-view. The master reference for comparing against is the specified behavior of the Instruction Set Architecture (ISA). For example, an add instruction that adds two register and places the result in another register must be validated for all specified register combinations as well as all register values. Since exercising all possible combinations would create an infinitely long test code, the difficultly of this task is heuristically deriving tractable test code that covers the significant state transitions.

In addition, the processor requires a coherent memory hierarchy. The physical memory hierarchy may store the same address contents in multiple locations, such as in the cache and in main memory, but these contents must be consistent. The memory hierarchy must also provide a minimum amount of memory management, such as trapping illegal memory accesses, to prevent the processor from locking up. Hence, the physical memory hierarchy must be verified to be logically correct over a wide range of possible operations.

In collaborating with ARM Ltd. on this research project, both the ARM8 processor core behavioral model and its suite of validation test code was received from ARM Ltd. [arm97]. By restricting any changes to the behavior model, the validation suite could be used verbatim for verifying the processor core and its implementation of the ARM v4 ISA. Test code had to be developed for the rest of the microprocessor and external system components.

The basic design flow to create this test code is shown in Figure 4.17. This verification flow ran in conjunction with the high-level behavioral design once enough of the system was specified to be simulatable. Individual code was developed to test

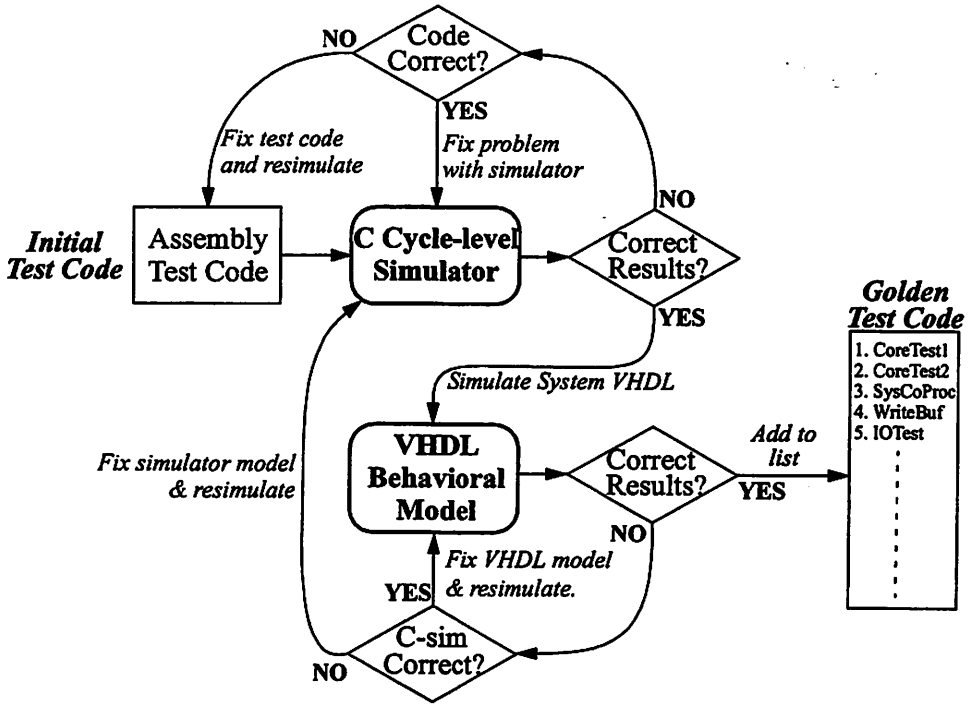


FIGURE 4.17 : Behavioral Verification Flow for Developing Test Code Suite.

each major part of the microprocessor (e.g. System Coprocessor, Cache Controller, Write Buffer, etc.) as well as the external SRAM chip and I/O chip. To guarantee correctness by design, the code was written to be self-checking; the result of any operation under test would be compared against the expected value, and flag an error on a mismatch. Both the cycle-level and VHDL simulator models included basic I/O so that error messages could be printed to the simulator screen.

The self-checking test code enforces consistency between the C simulator and the ISA, as well as consistency between the C simulator and VHDL behavioral model. The golden test code suite consists of: 28 programs from ARM Ltd. that test individual ARM8 blocks, 32 programs from ARM Ltd. which comprise their validation suite to verify the entire ARM8 core, and an additional 10 programs that were developed for the remainder of the system.

Once all the test code successfully ran on the VHDL system simulation, there was extremely high confidence in the functional correctness of the behavioral model. The additional step taken was to boot the operating system on the VHDL simulation to

further verify functionality.

4.5.2 Test Vector Generation

Once the golden test code suite was developed, test vectors for individual blocks could be generated from a behavioral VHDL simulation as demonstrated in Figure 4.18. The block under test (BUT) has its pins traced while running the test code that exercises the BUT. The output waveform database is then sampled and converted to time-independent test vectors via the leap2crf script. Each vector corresponds to a single clock phase, gives input state at the beginning of the phase, and indicates what the output state should be at the end of the phase.

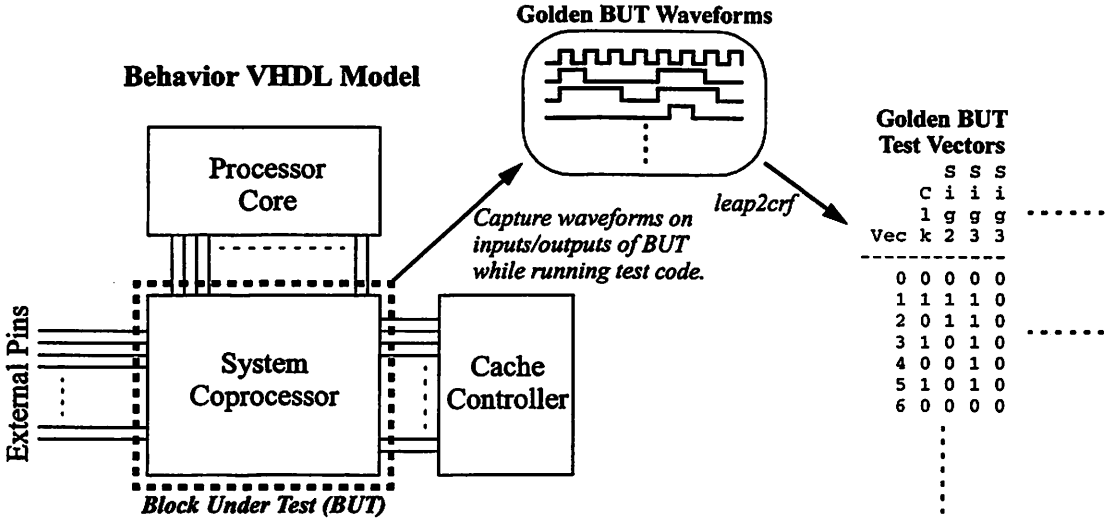


FIGURE 4.18 : Automatic Test Vector Generation

A header file describes the direction of each pin to be input, output, or bidirectional. In the test vector file, a bidirectional signal is treated either as an input or output for any given test vector, depending on whether the signal was actively driven by the BUT in that test vector or not. An *H* or *L* state indicates a driven input, while a *1*, *0*, or *Z* state indicates an output signal. A skew file provides a skew number for each signal, which can be used to model setup and hold time constraints on the BUT.

The golden test vectors are used verbatim as structural VHDL test vectors. The crf2epic script allows switched-level simulation test vectors to be automatically

generated from the golden test vectors. This script takes a time base as an argument so that simulation test vectors can be created at arbitrary cycle times allowing vector to be quickly generated for multiple voltages. When setup and hold time constraints had to be renegotiated between blocks, the blocks' skew files were modified and adjusted switched-level simulation test vectors could be automatically generated.

A hierarchal set of test vectors were generated. Some vectors were for smaller blocks, such as an individual ARM8 macro block or cache controller, and others were for larger blocks, such as the entire ARM8 core, the cache subsystem, and even the entire microprocessor chip. This allowed the design to be verified at all levels of the hierarchy. Most of the logical debugging occurred within the smaller blocks, while verifying the larger blocks ensured the design was connected properly.

4.5.3 Structural Simulation

The script genTB was used create a testbench for a given block. The block's VHDL entity provides the necessary pin-direction information, and creates a VHDL stimulus file which reads the test vector file. At the beginning of each vector, the inputs are toggled as specified, and at the end of the specified half-cycle time, the outputs are checked against the expected outputs in the test vector file. If any errors are present, an error flag is issued giving the exact location of the error.

As the design was refined from a high-level behavioral specification to a lower-level structural VHDL specification, the test-bench simulation was run for each block to ensure proper functionality. If all test vectors passed successfully, the block was deemed to be functionally correct and ready for physical implementation.

4.5.4 Transistor Netlist Simulation

Timemill was the tool of choice for transistor level simulation. The crf2epic script generates two files, an input vector file and an output vector check file. The input

vector file specifies input signal transitions at regular time intervals, equal to the desired half-cycle time. The skew file can be used to shift signal transitions around with respect to the clock edge. The output vector check file specifies what the output pin states should be at regular intervals. Again, with the skew file, when the outputs should be stable with respect to the clock edge can be varied with setup-time constraints.

The initial simulation is run on a block's schematic netlist, to ensure topological functionality. This step catches logical transistor design errors. Once the block's layout is complete and verified back against the schematic via LVS, the layout was extracted to create a netlist. Timemill was run on this new netlist, and would catch setup and hold time violations for the block due to the interconnect parasitics.

4.6 Timing Verification

There are two parts to timing verification: race-condition analysis for functionality, and critical-path analysis for performance. Pathmill was extensively used for timing verification, in which the key to speeding up the analysis turn-around was a schematic design that followed a simple naming methodology. Then, Pathmill could be scripted to quickly find the important paths to be analyzed.

Timing was first run on the schematic design to flag potential race paths as well as identify critical paths based upon logic depth. Any path longer than 30 gate delays was reduced. Paths in the range 24-30 gates deep were reduced if a simple schematic fix was possible, since they had a high probability of becoming critical with extracted parasitics. Any path depth less than four was increased, which would occur as the unintended side-effect of a sped-up latch. Finally, any path that could induce a clock glitch (e.g. an output signal from a Phi2 latch gating a Phi2 clock driver) was flagged and fixed.

Once the first-pass layout was complete, Pathmill was run again on the

extracted netlist. Multiple timing iterations were used to fix critical paths that cropped up due to lack of interconnect capacitance in the schematic netlist. The labelSpice script was written to ensure that the layout netlist had the exact node names as the schematic netlist. Hence, the exact same timing input deck could be used both on the schematic and extracted netlists.

4.6.1 Schematic Naming Methodology

All cell instances that either maintain state (e.g. latches, flip-flops, registers) or are clock drivers have specific naming requirements. The name conveys information on the type of state and on which phase, 1 or 2, the instance is active. Within the master instance, the actual state nodes also require specific names. Then, for example, through simple pattern matching all phase-2 latch nodes can be specified in one statement.

The special instance labels are:

1. *p{1,2}lat{#}* Phase-1/2 transparent latch.
2. *p{1,2}flop{#}* Phase-1/2 edge-triggered flip-flop.
3. *p{1,2}clk{#}* Phase-1/2 clock driver.
4. *p{1,2}enab{#}* Phase-1/2 enabled signal (tri-stated).
5. *p{1,2}pre{#}* Phase-1/2 pre-charged node.

The {#} is used to differentiate multiple instances of the same type/phase on a given schematic sheet (e.g. *p1lat0*, *p1lat1*, etc.). In addition, there is the special phase designator, *A*, indicating an asynchronous signal (e.g. *pAenab0*).

The special state node labels are:

1. *epic_latch* Latch node between pass gate and cross coupled inverters.
2. *epic_latch{1,2}* The two latch nodes in a flip-flop.
3. *epic_clock* Clock node right after the enabled gate cell in the clock drivers.
4. *epic_enable* Node immediately preceding the driving inverter on a tri-state bus. Inverter enabling signal must be the output of a clock driver.
5. *epic_pre* Precharged node. Precharge signal must be the output of a clock driver.
6. *epic_register* Latch node within a register file. Labelled differently because it requires special handling since the register files are bi-phase.

In addition, any one of the above state nodes can be suffixed with a letter indicating parallel nodes on a single schematic sheet (e.g. *epic_latcha*, *epic_latchb*).

Through wild-card matching, all phase-2 latch nodes are simply **p2lat*.epic_latch**. To ensure all instances are properly labelled, a script was written to parse through the netlist looking for any node names not associated with a labelled instance, and flag them to be changed. Likewise, the inverse can be checked as well. However, if both the instance and node labels are omitted, then the node cannot be caught just from parsing the netlist. However, it will most likely show up as a critical path, and can be changed after the initial timing run. Thus, this methodology proved very robust in properly annotating all state nodes.

4.6.2 Path Identification

When performing timing analysis, all delay measurements must be referred back to a common signal, typically the output of the global clock driver. As was demonstrated in Section 4.3.4, the maximum skew between any two clock driver outputs is less than 4% of the cycle time. Because all the state nodes (except *epic_register*) only change state upon a clock driver output transition, timing between

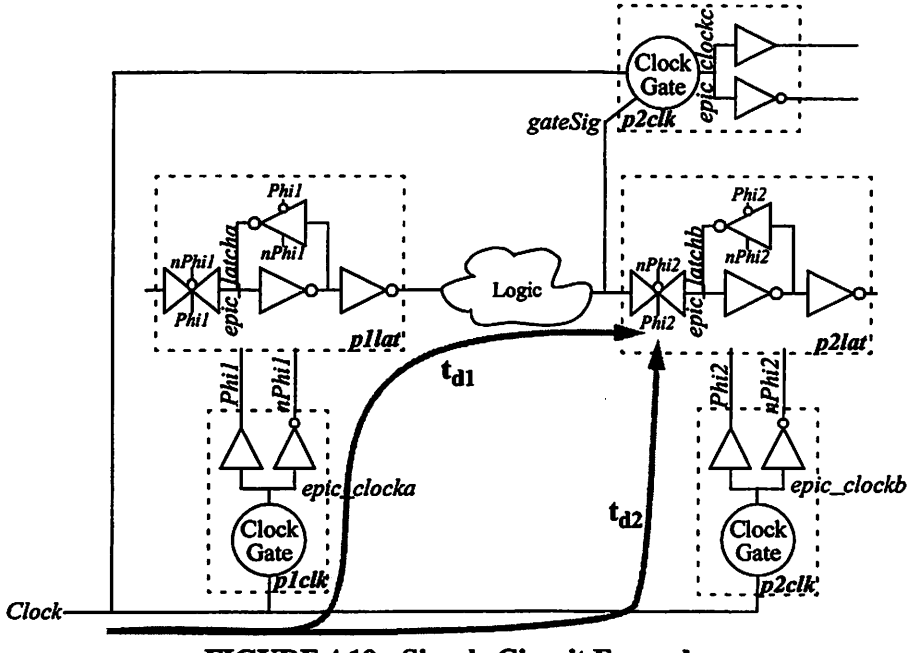


FIGURE 4.19 : Simple Circuit Example.

state nodes can ignore delay through the clock drivers, with a 4% margin of error.

In Figure 4.19, when *Clock* goes low, *Phi1/nPhi1* get asserted, latching a new data value onto the state node, *epic_latcha*. When *Clock* goes high, *Phi2/nPhi2* get asserted, latching a new value onto the state node *epic_latchb*, at which time, the input to *p2lat* must be stable. To ensure this, the delay through the logic block, t_{CRIT}, referred back to the common *Clock* signal must be less than the target cycle time:

$$t_{CRIT} = t_{d1} - t_{d2} = t_{Clock \rightarrow epic\_latcha} + t_{epic\_latcha \rightarrow epic\_latchb} - t_{Clock \rightarrow epic\_latchb} \quad \text{(EQ 4.9)}$$

where the delay through the clock buffers is:

$$t_{Clock \rightarrow epic\_latch\{a,b\}} = t_{Clock \rightarrow Phi\{1,2\}} + t_{Phi\{1,2\} \rightarrow epic\_latch\{a,b\}} \quad \text{(EQ 4.10)}$$

The first component differs at most by the maximum clock skew. The second component will be essentially the same between the two because the output of the clock drivers have nearly identical rise/fall times. Hence, Equation 4.9 can be rewritten as:

$$t_{CRIT} = t_{epic\_latcha \rightarrow epic\_latchb} + \delta_{ClkSkew} \cong t_{epic\_latcha \rightarrow epic\_latchb} \quad \text{(EQ 4.11)}$$

Hence, timing path analysis only requires looking between any two state nodes (latch, flip-flop, precharged, and enabled), which is virtually equivalent to referring the path calculation back to a common point. There are two exceptions to this rule.

The first exception is for clock gate signals. For the path from *p1lat* to *p2clk*:

$$t_{CRIT2} = t_{Clock \rightarrow epic\_latcha} + t_{epic\_latcha \rightarrow gateSig} + t_{gateSig \rightarrow epic\_clock} - t_{Clock \rightarrow epic\_clock} \quad \text{(EQ 4.12)}$$

The last term is dropped, because the *epic_clock* node should be stable before *Clock* changes. Equation 4.12 can be reduced to:

$$t_{CRIT2} = t_{Clock \rightarrow epic\_latcha} + t_{epic\_latcha \rightarrow epic\_clock} \quad \text{(EQ 4.13)}$$

Thus, the delay between the state node (*epic_latcha*) and the clock node (*epic_clock*) must be less than the target cycle time minus the delay through the clock buffer. The

mean delay through the clock buffer is 8% of the half-cycle time of over voltage. Thus, these paths have a shorter time to complete.

The other exception are *epic_register* nodes. All the register files are bi-phase, so the standard clock drivers cannot be used. Instead, custom clocking circuitry is required, which while designed to match the clock driver as well as possible, they do not meet the same skew tolerance. Thus, input and output to these nodes must be given an extra margin of tolerance, and carefully simulated to ensure no race condition exists.

4.6.3 Timing Analysis

Path delay varies from VCO delay in one of two ways. First, there are paths whose delay is dominated by PMOS devices (since $V_{TP} > V_{TN}$) and/or interconnect capacitance, and they slow down at low voltage with respect to the VCO. The second class of paths, whose delay is dominated by gate/diffusion capacitance (which increases with voltage) and/or interconnect RC delay, slow down at high voltage with respect to the VCO. A typical commercial design is only concerned about a singular voltage operating point, but DVS must operate over a broader range of voltage.

However, DVS only requires timing analysis at two voltages, which are the extremes of the desired operating range. For the prototype processor, the voltages are 3.3V and 1.2V. If timing constraints are met at these two points, then the timing constraints will be met at all intermediate points in between.

Using the schematic naming methodology, a complete timing analysis can be performed with 16 individual Pathmill runs, listed in Table 4.6. This analysis checks for short paths, long paths, and illegal paths. The target cycle time is 10ns at 3.3V, and 80ns at 1.2V.

Checks 1-4 analyze paths between different-phase state nodes, so they only have a half-cycle time to complete. In addition, the maximum delay is further reduced

TABLE 4.6 Timing Analysis Pattern Set

#	Source	Sink	Maximum Delay (ns)	
			V_{DD}=3.3	V_{DD}=1.2
1	*p1{lat,flop,enab,pre}.epic_**	*p2{lat,enab}.epic_**	4.5 (s)	36 (s)
2	*p2{lat,flop,enab,pre}.epic_**	*p1{lat,enab}.epic_**		
3	*p1{lat,flop,enab,pre}.epic_**	*p2{flop,pre}.epic_**	4.5 (h)	36 (h)
4	*p2{lat,flop,enab,pre}.epic_**	*p1{flop,pre}.epic_**		
5	*p1{lat,flop,enab,pre}.epic_**	*p1flop.epic_latch1*	9.5 (h)	76 (h)
6	*p2{lat,flop,enab,pre}.epic_**	*p2flop.epic_latch1*		
7	*p1{lat,flop}.epic_**	*p2clk.epic_clock*	4.0 (h)	32 (h)
8	*p2{lat,flop}.epic_**	*p1clk.epic_clock*		
9	*p1{lat,flop,enab,pre}.epic_**	*epic_register*	4.0 (h)	32 (h)
10	*p2{lat,flop,enab,pre}.epic_**			
11	*epic_register*	*p1{lat,flop,enab,pre}.epic_**	4.0 (h)	32 (h)
12		*p2{lat,flop,enab,pre}.epic_**		
13	*p1{lat,flop,enab,pre}.epic_**	*p1clk.epic_clock*	***Should never occur*** (same phase gate signal)	
14	*p2{lat,flop,enab,pre}.epic_**	*p2clk.epic_clock*		
15	*p1{lat,flop,enab}.epic_**	*p1pre.epic_pre*	***Should never occur*** (same phase pre input)	
16	*p2{lat,flop,enab}.epic_**	*p2pre.epic_pre*		

by 5% of the cycle-time to provide margin for clock skew. The first two checks are soft (s) constraints, because the sink state node is transparent, and will continue to operate if the maximum delay is exceeded. Checks 3-4 are hard (h) constraints, because a functional error will result if the maximum delay is exceeded. Checks 5-6 have a full cycle to complete minus 5% margin for clock skew. Checks 7-8 provide 10% margin to account for both clock skew and delay through the clock buffer. The register checks (9-12) analyze both input/output paths, and provide 10% margin to give additional headroom for the custom clock drivers. The last four checks (13-16) check for illegal paths that should never occur.

The output from Pathmill provides an ordered list of long and short paths for

```
  path # of
 delay stages              from node          =>          to node
=========================================================================
1 4.888  20   I71.I17.I4.p1lat0<13>.epic_latch   I71.p2lat2<1>.epic_latch
2 4.872  20   I71.I17.I4.p1lat0<13>.epic_latch   I71.p2lat3<1>.epic_latch
3 4.865  18   I72.I1.I4.p1pre<0>.epic_pre        I72.p2lat8<1>.epic_latch
```

FIGURE 4.20 : Example Pathmill Timing Analysis Output

each check, as shown in Figure 4.20. In addition, Pathmill reports each stage in these paths, such that they can be quickly identified in the schematic. If any paths exceed the maximum delay check, they are fixed through schematic changes, and the design is re-analyzed.

5 Architectural Design Methodology

While it is important to be energy conscious at all levels of the design hierarchy, energy-efficiency optimizations at the level of architectural design generally yield the biggest gains. The closer the design approaches to the final physical implementation, the smaller the gains get because the scope of possible optimizations narrows.

Unfortunately, traditional architectural design methodologies for microprocessor systems focus primarily on performance, with energy consumption considered as an afterthought. This approach is slowly evolving to consider energy consumption earlier in the design process, but the radical change in design methodology that is required has yet to happen. This wholesale change requires the incorporation of energy estimation into the high-level system simulator (Section 4.2), so that both performance and energy consumption can be estimated at the highest level of the design space. This allows for architectural design choices to be evaluated for their overall impact on system energy efficiency, and not just strictly performance.

The first section describes the architectural design and methodology of the processor system, while subsequent sections describe in more detail the major components of the microprocessor itself -- the processor core, the cache system, and the system-control coprocessor.

5.1 System Architecture

Microprocessor systems generally resemble the generic architecture shown in Figure 5.1. The processor bus connects the microprocessor to the main external memory (ROM, RAM), input/output devices via I/O controllers, and peripheral subsystems via bus controllers. A PAL or PLD is typically used to generate the control signals between the various chips.

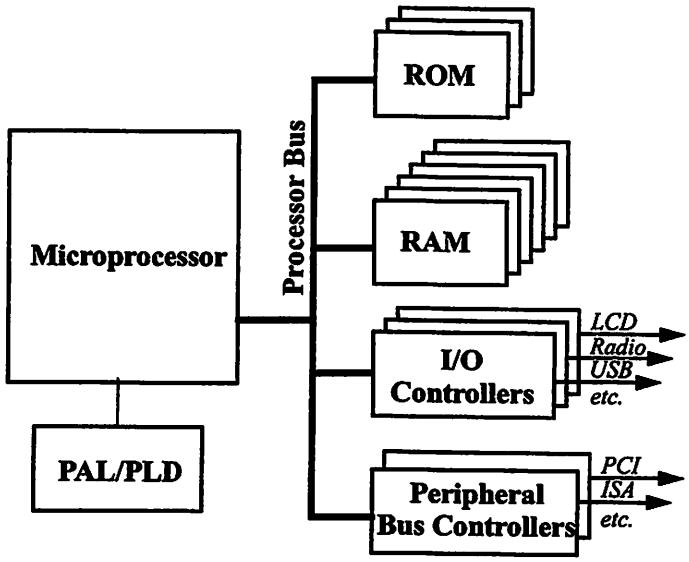


FIGURE 5.1 : Generic Microprocessor System Architecture.

While the generic system appears to have little room for optimization, a number of architectural design choices are available which can significantly impact performance and energy consumption. This is beyond the scope of this work, however, since in trying to demonstrate DVS at the system level, constraints had to be placed onto the system organization as described below.

5.1.1 Modifications for DVS

A key modification to implement DVS at the system level is a DVS-compatible processor bus, which requires the bus to operate across the entire range of operating voltage and performance. Existing processor bus specifications do not support this, so a custom solution was developed for the processor bus. This, in turn, required custom

chips to communicate on the bus, since no available commercial chip supports this custom bus implementation.

To reduce the number of chips requiring a custom implementation, the system was organized as shown in Figure 5.2. In this architecture, only three chips require custom implementation: the microprocessor, the external RAM, and the I/O interface chip. For the targeted application of portable electronic systems, this is a reasonable solution since the bulk of the bandwidth on the processor bus is between the microprocessor and the main memory. The I/O bandwidth is in the range of 10 kB/sec to 5 MB/sec for each I/O device, which is a small fraction of the available 200 MB/sec peak bandwidth on the processor bus.

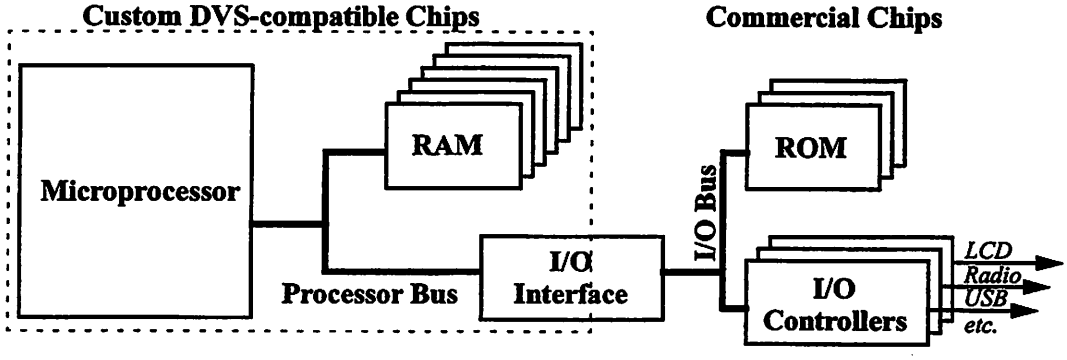


FIGURE 5.2 : Prototype System Architecture Incorporating DVS.

The PAL/PLD was eliminated for two reasons. First, commercial PAL/PLDs are not DVS-compatible. Second, and most important, they consume an inordinate amount of power in a typical embedded processor system given the functionality that they provide. As such, this functionality (memory controller, interrupt controller, etc.) was implemented on-chip within the system-control coprocessor, providing a significant reduction in system energy consumption.

5.1.2 Cache Benefits and Limitations

On-chip cache is essential for providing good system performance and energy efficiency in a general-purpose processor system. An on-chip cache access is simply

much faster than an off-chip access to main memory. While cache accesses can typically be achieved at the processor clock rate, external accesses need to traverse through the processor's external interface, the external system bus, and the memory chip's bus interface, all of which increase access latency anywhere from a few processor cycles to tens of cycles [henn95].

A cache access consumes less energy than an access to main memory because the same things that slow the access down -- external bus interfaces and the external bus itself -- also increase its energy consumption. For the prototype system, a cache access is only 100 pF/access, while an external memory access is 500 pF/access, yielding a 5x reduction in energy consumption per memory access.

However, cache misses require transferring several words into a cache line from main memory, and an equal amount of words need to be transferred from the cache back to main memory when the cache line being replaced is dirty. So what is important to evaluate is the average capacitance/access for a memory access to the on-chip cache, including the effects of cache misses:

$$C_{MA|AVE} = (1 - MR) \cdot C_{CACHE} + MR \cdot LS(C_{CACHE} + C_{EXTMEM}) \cdot (1 + DR) \qquad \text{(EQ 5.1)}$$

where MR is the fractional cache miss rate, C_{CACHE} is the capacitance of a cache access (100 pF/access), C_{EXTMEM} is the capacitance of an external memory access (500 pF/access), LS is the line size in words, and DR is fractional amount of cache misses that are also dirty. From Equation 5.1, we can calculate what is the maximum miss rate that will ensure $C_{MA|AVE} < C_{EXTMEM}$ so that adding the cache actually improves system energy efficiency. From the prototype system, $LS = 8$ and $DR = 10\%$, such that as long as $MR < 7.7\%$, then $C_{MA|AVE} < C_{EXTMEM}$. This level of miss rate can be achieved with a cache size as small as 2kB [henn95].

So, for almost all practical sizes of an on-chip cache, the inclusion of the cache will not only improve processor performance by reducing the average memory access

time, but it will also improve the overall system energy efficiency. The cache used in the prototype design has only a 0.5% miss rate (derived from benchmark program simulations), reducing $C_{MA|AVE}$ to only 125 pF/access, which is still 4x lower than C_{EXTMEM}.

The primary drawback to using a cache is its non-deterministic behavior, which can adversely impact real-time systems. This can be addressed in one of two ways. A software solution entails that any time-critical operation take into account a worst-case latency assuming cache misses. If this is not feasible, a hardware solution consists of either dedicated on-chip memory (e.g. ROM, SRAM) separate from the cache, or a cache which allows specified cache lines to be locked into the cache.

For the PDA-like applications running on the prototype demonstration system, a hardware solution to ensure operational latency was not necessary, as the latency issues were sufficiently dealt with in software. Thus, implementing the on-chip cache was essential for improving system energy efficiency, yielding the simplified microprocessor architecture shown in Figure 5.3. The system control block incorporates the glue logic required to seamlessly connect together the custom chips of the prototype system. These three components are described in further detail in Section 5.2-4.

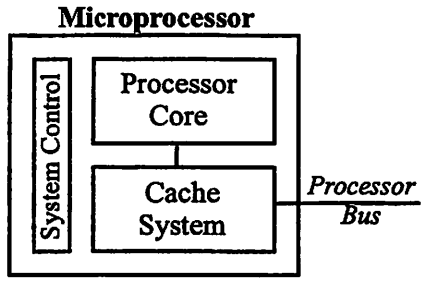


FIGURE 5.3 : Prototype Microprocessor Architecture.

5.1.3 Main Memory Architecture & Processor Bus Topology

Commodity SRAMs and DRAMs used for main memory are typically eight bits wide. This data width has been used since the earliest microprocessor days when the processor itself was only eight bits wide. While the bus width of the microprocessor has

increased to improve memory bandwidth to the processor, the width of the RAM chips has remained unchanged. To increase the bandwidth of the memory bank, multiple memory chips are enabled in parallel, as shown in Figure 5.4. In this example, four 8-bit RAM chips are accessed simultaneously to provide a 32-bit data word to the processor.

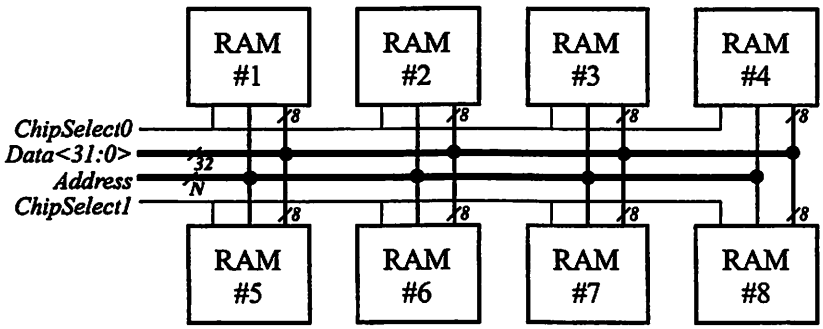


FIGURE 5.4 : Typical Main Memory Architecture.

While this approach successfully meets the bandwidth (performance) demand, it increases the energy consumed per access. However, a 32-bit RAM chip would require only one memory chip to be activated per access, eliminating the unnecessary energy consumption of the other three chips. The primary drawback to a larger bus width is increased pin count on the memory chip. A typical 8-bit wide, 16Mb RAM chip has a total of 29 address and data pins. If the address and data are multiplexed onto the same bus, then only five additional pins are required to support 32-bit accesses. The proposed architecture is shown in Figure 5.5.

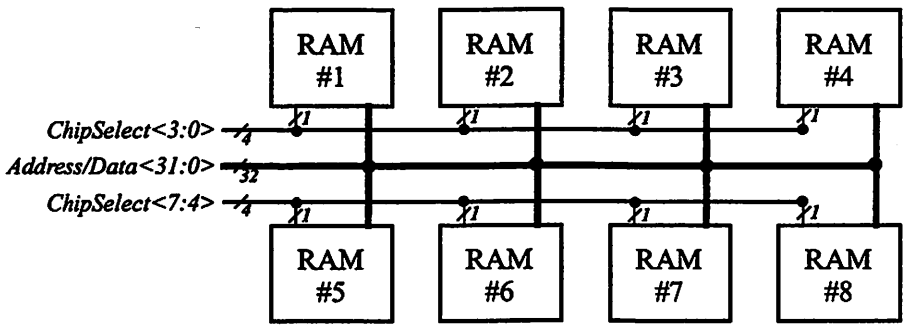


FIGURE 5.5 : Proposed Main Memory Architecture.

Two bus cycles are required for a single-word memory access in this multiplexed approach. However, for a processor with a cache, most memory accesses

are cache line reloads. Thus, if the processor bus can support burst-mode accesses, then the cost of placing the address on the multiplexed bus can be amortized over multiple data words. For a cache-line length of eight, the effective bandwidth is 8/9, or 89% of peak capacity. Benchmark simulation demonstrated that with a large 16kB cache, external accesses are predominantly cache-line reads and writes such that actual bus capacity is only 15% less than peak capacity. This slight degradation in performance is more than compensated for by the reduction in energy consumption of the external memory chips. Thus, the processor bus was designed to support burst accesses, as described in further detail in Section 7.4.

5.1.4 I/O Considerations

The I/O interface chip is essentially a bridge between the processor bus and the I/O bus, such that its implementation is relatively simple. The critical functionality required is flow control between the processor and the autonomous I/O controllers, which all operate at different clock frequencies. This is provided through a simple state machine on the chip in conjunction with wait signals which can halt either the processor, or an I/O controller, as necessary.

To improve the system energy efficiency, the I/O interface chip also allows for packed I/O data writes. Typical I/O data transfers are one byte wide, such that if I/O data is transferred in single bytes, two processor bus cycles are required to complete the data transfer, one for the address and one for the data byte. The aggregate I/O data transfer rate is 0.5 bytes/cycle. However, by allowing four bytes to be packed into a single word, then the transfer rate increases to 2 bytes/cycle. The I/O interface chip is responsible for unpacking the word into four individual byte writes to the addressed I/O controller. Packing I/O byte reads is not done, since the processor would have to wait until the fourth read before it could process the data, adding additional latency to the I/O transfer.

To improve system performance, the I/O interface chip also supports direct memory accesses (DMA) to allow I/O controllers to directly access the main memory. This frees up processor cycles that would otherwise be required since the processor must act as an intermediary between the I/O controllers and main memory when DMA is not supported.

In the prototype system, this functionality is performed by the virtual I/O controller (Section 7.8) which eliminates the need to implement this added functionality in silicon. The actual I/O interface chip that was custom implemented only provides the voltage level-conversion and signalling required to support I/O transfers and DMA between the processor bus and the virtual I/O controller, which was implemented with a Xilinx FPGA and a StrongArm processor system. However, for a complete system solution, this functionality must be implemented on the custom I/O interface chip. Since the average number of I/O accesses per processor cycle is well below one, the energy consumed by this full-custom I/O interface chip implementation is negligible with respect to the overall processor system energy consumption.

5.2 Processor Core

The design of the processor core presents a variety of opportunities for improving the overall microprocessor system's energy efficiency. To facilitate the design of the prototype system in Chapter 7, an ARM8 behavioral model from ARM Ltd. was incorporated into the high-level design specification of the prototype processor. While significantly speeding up the implementation, this model also constrained the processor design space by fixing the instruction set architecture as well as the core microarchitecture.

As such, this section explores common processor core optimizations for improving performance and/or reducing power dissipation. To properly evaluate these optimizations, it is imperative to analyze the system-level performance and energy

consumption improvement, rather than focus only on the improvement within an individual processor block, as is often done. Those optimizations that demonstrate to improve the overall system energy-efficiency may be utilized in the design of a future, more energy-efficient microprocessor.

5.2.1 Instruction Set Architecture

Typically, an instruction set architecture (ISA) is designed solely with performance in mind. High-level performance simulators allow the architect to explore the ISA design space with reasonable efficiency. Energy is not a consideration, nor are there high-level simulators available to even let the architect estimate energy consumption. Simulation tools exist, but require a detailed description of the microarchitecture so that they are not useful until the ISA has been completely specified. Processors targeted towards portable systems should have their ISA designed for energy efficiency, and not just performance.

Many processors have 32-bit instruction-words and registers. Register width generally depends on the required memory address space, and cannot be reduced; in fact, more recent microprocessors have moved to 64 bits. For low-energy processors, 16-bit instruction widths have been proposed. Static code density can be reduced by 30-35%, while increasing the dynamic run length by only 15-20% over an equivalent 32-bit processor [bund93a][arm95]. Using 16-bit instructions reduces the energy cost of an instruction fetch by up to 50% because the size of the memory read has been halved [bund94]. In system's with 16-bit external busses, the advantage of 16-bit instructions is further widened [free94][arm95]. Since instruction fetch consumes about a third of the processor's energy [burd94b][mont96b], total energy consumption is reduced by 15-20%, which is cancelled out by the 15-20% reduction in performance, giving approximately equivalent energy efficiency.

The available data indicates that this technique significantly improves energy

efficiency only if the external memory's energy consumption dominates the processor's energy consumption, or if the external processor bus is 16 bits, instead of the more typical 32 bits. However, this is only true for no, or very small, on-chip caches (< 8kB). Since most microprocessors today contain at least a moderately-sized on-chip cache (16kB or larger) enabled by CMOS VLSI process scaling, utilizing 16-bit instructions will typically have negligible impact on microprocessor energy efficiency. It is only useful for reducing external memory requirements in cost-sensitive system designs.

The number of registers can be optimized for energy efficiency. The register file consumes a sizable fraction of total energy consumption since it is typically accessed multiple times per cycle (10% of the total energy in [burd94b], as well as in the prototype system). In a register-memory architecture, the number of general purpose registers is kept small and many operands are fetched from memory. Since the energy cost of a cache access surpasses that of a moderately sized (32) register file, this is not energy efficient. The other extreme is to implement register windows which is essentially a very large (100+) register file. The energy consumed by the register file increases dramatically increasing total processor energy consumption 10-20%. Unless this increase in energy is compensated by an equivalent increase in performance, register windows are not energy efficient. One study compared register files of size 16 and 32 for a given ISA, and found that for 16 registers, the dynamic run length is 8% larger [bund93b]. The corresponding decrease in processor energy due to a smaller register file is on the order of 5-10%. There appears to be a broad optimum on the number of registers since the energy efficiency is nearly equal for 16 and 32-element register file.

The issue of supported operation types and addressing modes has been a main philosophical division between the RISC and CISC proponents. While this issue has been debated solely in the context of performance, it can also have an impact on energy consumption. Complex ISAs have higher code density, which reduces the energy

consumed fetching instructions and reduces the total number of instructions executed. Simple ISAs typically have simpler data and control paths, which reduces the energy consumed per instruction, but there are more instructions. These trade-offs need to be analyzed when creating an ISA.

The amount of hardware exposed (e.g., branch delay slot, load delay slot, etc.) is another main consideration in ISA design. This is typically done to improve performance by simplifying the hardware implementation. Since the scheduling complexity resides in the compiler, it consumes zero run-time energy while the simplified hardware consumes less energy per operation. Thus, both the performance is increased and the energy/operation is decreased, giving a two-fold increase in energy efficiency. A good example of radically exposing the hardware architecture are very long instruction word (VLIW) architectures, which will be discussed in more detail in the following section.

5.2.2 Architectural Concurrency

The predominant technique to increase energy efficiency in custom DSP ICs (fixed throughput) is architectural concurrency; with regards to processors, this is generally known as instruction-level parallelism (ILP). Previous work on fixed throughput applications demonstrated an energy-efficiency improvement of approximately N on an N-way parallel/pipelined architecture [chan92]. This assumes that the instructions being executed are fully vectorizable, that N is not excessively large, and that the extra delay and energy overhead for multiplexing and demultiplexing the data is insignificant.

Moderate pipelining (4 or 5 stages), while originally implemented purely for speed, also increases energy efficiency, particularly in RISC processors which operate near one cycle-per-instruction. Energy efficiency can be improved by a factor of two or more [gonz95], and is essential in an energy-efficient processor.

5. 2. 2. 1 Superscalar Architectures

More recent processor designs have implemented superscalar architectures with parallel execution units, in the hope of further increasing the processor's execution concurrency. However, an *N*-way superscalar machine will not yield a speedup of *N*, due to the limited ILP found in typical code [john90][wall93]. Therefore, the achievable speedup will be less than the number of simultaneous issuable instructions and yields diminishing returns as the peak issue rate is increased. The speedup has been shown to be between two and three for practical hardware implementations [smit89].

If the instructions are dynamically scheduled in employing superscalar operation, as is generally done to enable backwards binary compatibility, the effective switched capacitance per cycle of the processor, C_{CPU}, will increase due to the implementation of the hardware scheduler. Also, there will be extra capacitive overhead due to branch prediction, operand bypassing, bus arbitration, etc. There will be additional capacitance increase because the *N* instructions are fetched simultaneously from the cache and may not all be issuable if a branch is present. The capacitance switched for un-issued instructions is amortized over those instructions that are issued, further increasing C_{CPU}.

The energy-efficiency increase can be analytically modeled. Equation 5.2 gives the *ETR* ratio of a superscalar architecture versus a simple scalar processor; a value larger than one indicates that the superscalar design is more energy efficient. The *S* term is the ratio of the throughputs, and the C_{CPU} terms are from the ratio of the energies, which is proportional to the effective switched capacitance since the architectural comparison is at constant supply voltage. The individual terms represent the contribution of the datapaths, C_{CPU}^{Dx}, the memory sub-system, C_{CPU}^{Mx}, and the dynamic scheduler and other control overhead, C_{CPU}^{Cx}. A 0 suffix denotes the scalar implementation, while a 1 suffix denotes the superscalar implementation. The quantity C_{CPU}^{C0} has been omitted, because it has been observed that the control overhead of the

scalar processor is minimal: $C_{CPU}{}^{C0} << C_{CPU}{}^{D0,M0}$ [burd94b].

$$ETR|_{RATIO} = \frac{S(C_{CPU}{}^{D0} + C_{CPU}{}^{M0})}{(C_{CPU}{}^{C1} + C_{CPU}{}^{D1} + C_{CPU}{}^{M1})} \qquad \text{(EQ 5.2)}$$

Simulation results show that $C_{CPU}{}^{C1}$ is significant due to control overhead and that $C_{CPU}{}^{M1}$ is greater than $C_{CPU}{}^{M0}$ due to un-issued instructions, thereby negating the *ETR* increase due to *S*. Since $C_{CPU}{}^{C1}$ increases quadraticly as the number of parallel functional units is increased, the largest improvement in energy efficiency would be expected for moderate amounts of parallelism. In this best case, however, the superscalar architecture yields no improvement in energy efficiency [gonz95].

5. 2. 2. 2 Superpipelined Architectures

These architectures also exploit ILP and offer speedups similar to those found in superscalar architectures [joup89], but their performance is lower because the number of stall cycles increases with the depth of the pipeline due to data dependencies. While these architectures do not need as complex hardware for the dynamic scheduler ($C_{CPU}{}^{Cx}$ is lower), they do need extra hardware for more complex operand bypassing ($C_{CPU}{}^{Dx}$ is higher). The net differences in speedup and capacitance should give superpipelined architectures an energy efficiency similar to superscalar architectures.

5. 2. 2. 3 VLIW Architectures

These architectures best exploit ILP by exposing the underlying parallelism of the hardware to the compiler's scheduler, which minimizes the complexity of the hardware. A good compiler is necessary to fully utilize the hardware. One such implementation from Multiflow gives a speedup factor, *S*, between two and six [lown93]. Because the parallelism is visible, VLIW processors do not require aggressive branch prediction, dynamic schedulers, and complex bus arbitration, so that the energy consumed per operation is roughly the same as for the scalar processor. The main additional energy cost is for the communication network that connects the

autonomous functional units that comprise the VLIW processor, and executing the instructions that shuffle data between them. Even with a pessimistic estimate of 50% for the energy per operation increase, the VLIW processor's energy efficiency increases anywhere from 33% to 300%.

5. 2. 2. 4 Summary

Superscalar and/or superpipelined architectures are commonly used today because of the increase in performance while maintaining backward machine code compatibility. Unfortunately, utilizing these architectures actually degrades processor energy efficiency. The most energy-efficient processor available today is a simple scalar design with a five-stage pipeline [mont96]. While VLIW architectures demonstrate very promising improvement in processor energy-efficiency, the required change in the ISA and the increased requirements on the compiler has severely limited their usage.

5.2.3 Microarchitecture

The processor control typically knows which pipeline stages are being used each cycle. Those pipeline stages not used in a given cycle should have their clock disabled for that cycle. This is particularly important to do in superscalar architectures that typically have only a fraction of the entire processor being utilized in any given cycle. With only a small overhead cost, this technique increases processor energy efficiency by 15-25% (assuming that 40-50% of the processor is disabled 40-50% of the time) [gary94].

To maximize the benefit of clock-gating, null or no-operation (NOP) instructions should be suppressed. In many microarchitectures, NOP instructions are mapped to real instructions. Although NOPs write to a null register, they consume more than half the energy of a normal instruction, as demonstrated by empirical measurements described in [tiwa96]. Instead, NOPs should be detected by a comparator

in the instruction decode stage, and later stages executing on the NOP should be disabled. Similarly, pipeline stalls and/or bubbles should not inject NOP instructions into the pipeline but should instead cause subsequent pipeline stages to be disabled during the appropriate cycle.

Correlation of data is often exploited for energy efficiency in signal processing circuits. While processors do not exhibit the same level of correlation as found in DSP circuits, high amounts of correlation can be found during effective address calculations, which are typically offset from a high-valued stack pointer. In most scalar processors, a single ALU calculates the effective addresses and all integer additions. By partitioning these two types of additions onto separate adders, the signal correlation increases by 16%, decreasing the adder's energy consumed per addition by an approximately equivalent 16%. Total processor energy efficiency can then be increased by 3-7%.

The *ETR* metric (Section 2.3.2) should be used to evaluate other microarchitectural design decisions for their relative impact on system energy efficiency. For those decisions with more than one feasible approach, the relative *ETR*s can be compared to select the most energy-efficient alternative.

5.2.4 Upper Bounds on Energy Efficiency

The bare essence of a processor is the ability to perform computational operations on data values. This capability, in its simplest form, is performed by the microprocessor's ALU, which is typically a very small fraction of the overall silicon area of the processor. In the prototype system, the fraction is well below 1%. All the surrounding circuit infrastructure merely enables the programmable nature of a general-purpose microprocessor.

Figure 5.6 plots the inverse of energy consumption, MIPS/Watt, for the prototype processor's most elementary component, the adder, and demonstrates how each added level of complexity required for implementing a complete microprocessor

increases its energy consumption. Energy consumption is measured at each complexity level, using switch-level circuit simulations, when the processor is executing a common sequence of code. The adder, or similarly the shifter, can operate at 278,000 MIPS/W at 1.2V in our 0.6µm process, and sets the absolute lower bound on energy consumption. Data storage for the adder or shifter's operands necessitates the need for a register file. Including a 30x32b register file increases energy consumption by 2.7x, yielding a more practical upper limit of 105,000 MIPS/W.

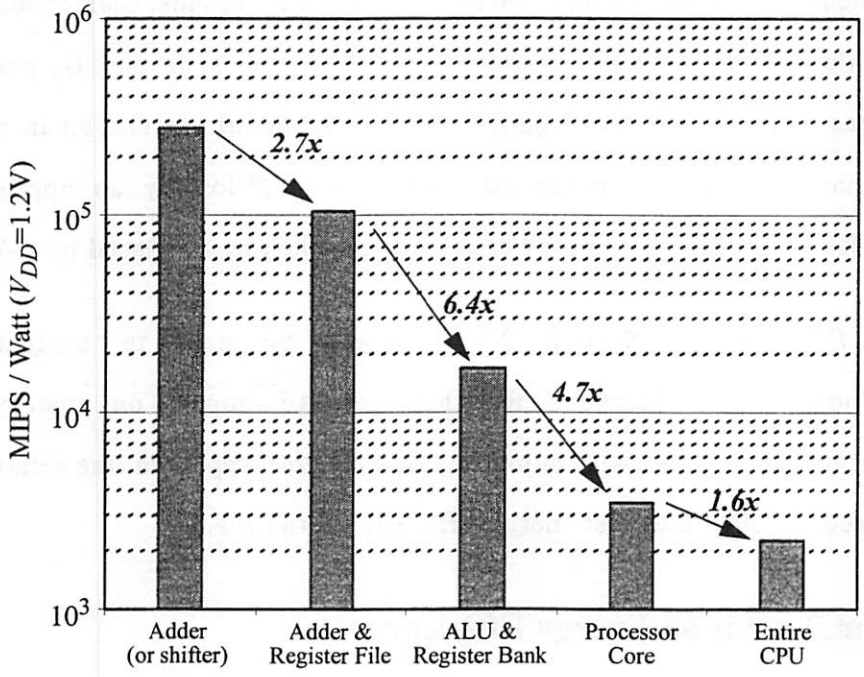


FIGURE 5.6 : Energy Consumption for the CPU and its Base Components (0.6µm CMOS).

The biggest jump in energy consumption, a factor of 6.4x, is attributable to the additional hardware required to build a fully programmable ALU and register bank. This hardware includes latches, muxes, bus drivers, clock drivers, and associated control circuitry. Additionally, the ALU includes a logic unit, a zero-detect unit, and a fast 4-bit shifter, while the register bank includes a 5b→32b register file decoder for each of its two read ports and one write port. The second biggest jump in energy consumption, a factor of 4.7x, is due to the additional hardware of the processor core. This hardware supports instruction fetches, branches, branch prediction, loads, stores,

and other ISA-specific functionality. The processor core that was implemented operates at only 3,500 MIPS/W.

Thus, the hardware overhead required to implement a fully-programmable microprocessor core increases energy consumption 30x above that required for just the base adder/shifter and register file circuits. While this overhead will always dominate the total processor energy consumption, there is still significant room for improving a processor's energy efficiency over that of conventional designs. However, what this entails is redesigning the processor core from scratch, starting with the ISA, and progressing down to the microarchitecture, while making design decision designs based not only on improved performance, but improved energy consumption as well. Measures for both performance and energy consumption are captured when utilizing the *ETR* metric to evaluate energy efficiency.

It appears in Figure 5.6 that the overhead required to turn a processor core into a fully-functional microprocessor chip remains significant, showing an energy consumption increase of 1.6x above the not fully-optimized processor core. However, the full chip design, and the cache system design in particular, has room for improvement beyond that demonstrated in the prototype system so as not to mitigate the overall microprocessor energy-efficiency improvement due to a more energy-efficient processor core. The limiting factor for not reducing the full chip's energy consumption further was, in fact, the processor core itself.

5.2.5 Low-Energy Idle Mode Enhancements

Microprocessors in most single-user system applications, such as notebook computers and PDAs, spend a majority of their time idling. Thus, it is essential for the microprocessor to provide power down modes to minimize the energy consumption while idling. However, to achieve the full benefit of these modes requires an energy-conscious operating system that utilizes them.

The design of the PowerPC 603 processor provides a good demonstration of useful power down modes [gary94]. A doze mode stops the processor from fetching instructions, but keeps alive snoop logic for cache coherency and the clock generation and timer circuits, reducing power dissipation by 6.3x for this mode. A nap mode disables the snoop logic, only keeping alive the timer logic, reducing power dissipation another 2.7x. Lastly, there is a sleep mode which only keeps alive the phase-locked loop (PLL) and clock. The power is reduced an additional 17%, while the processor can be up and running at full speed within ten clock cycles, and a cache flush. Further power reduction can be achieved by disabling the PLL in the sleep mode, which reduces power dissipation another 25x (for a total power reduction of 500x), but at the cost of several thousand cycles (up to 200μs) to return to full speed.

It is important to notice how much the PLL, which is found on most microprocessors, limits the reduction of idle energy consumption. Frequently turning off the PLL is not a viable approach due to the large overhead of restarting it. Techniques for improving the energy efficiency of PLLs in power down modes are needed.

A significant benefit of DVS is the lack of a PLL. The VCO within the voltage regulation loop is continuously running, but at a very low level of power dissipation, so that the microprocessor can restart with only one cycle of latency. In the prototype system, there was only a single power down mode implemented (Sleep mode), which reduced power dissipation 500x from the peak level, and provided a single-cycle wake-up time.

While most microcontrollers and some embedded processors have power down modes, only a few microprocessors have them. It is an important technique to include in energy-efficient processors. The actual energy savings, though, depends more on how well the operating system can utilize these modes.

5.3 Cache System

The high-level specification of the cache system was a full custom design in the prototype system, which enabled all aspects of the cache to be optimized for energy efficiency. As such, the optimizations presented in the following sections were fully validated in the implementation of the prototype system. Significant improvement in energy efficiency was demonstrated as the cache system was implemented with approximately half the average capacitance per cycle as the processor core. Typically, embedded processors' energy consumption is dominated by their on-chip caches.

5.3.1 Cache size

Increasing the on-chip cache size will always decrease the cache miss-rate, thereby decreasing the number of external accesses and reducing the average memory access time [henn95]. Thus, on-chip cache size is typically maximized, given die-size constraints, for performance considerations.

Maximizing cache size will increase the capacitance/access to the cache. However, this increase can be mitigated in two ways so that capacitance/access does not scale proportionally to cache size. By sub-blocking the cache (Section 5.3.2), the actual memory array size being accessed will remain constant, independent of cache size. Although the interconnect capacitance increases with cache size, a hierarchically buffered bus structure will limit the increase to scale approximately logarithmically. Thus, if buffers are judiciously utilized, the average capacitance/access of a memory access ($C_{MA|AVE}$) will continue to decrease with cache size past 256kB as demonstrated in Figure 5.7. If, however, the interconnect capacitance scales proportionally with cache size, then there is an optimal cache size, beyond which the average capacitance/ access increases with cache size.

In summary, maximizing the cache size will generally maximize system energy efficiency by providing the highest performance and the lowest energy/access. For the

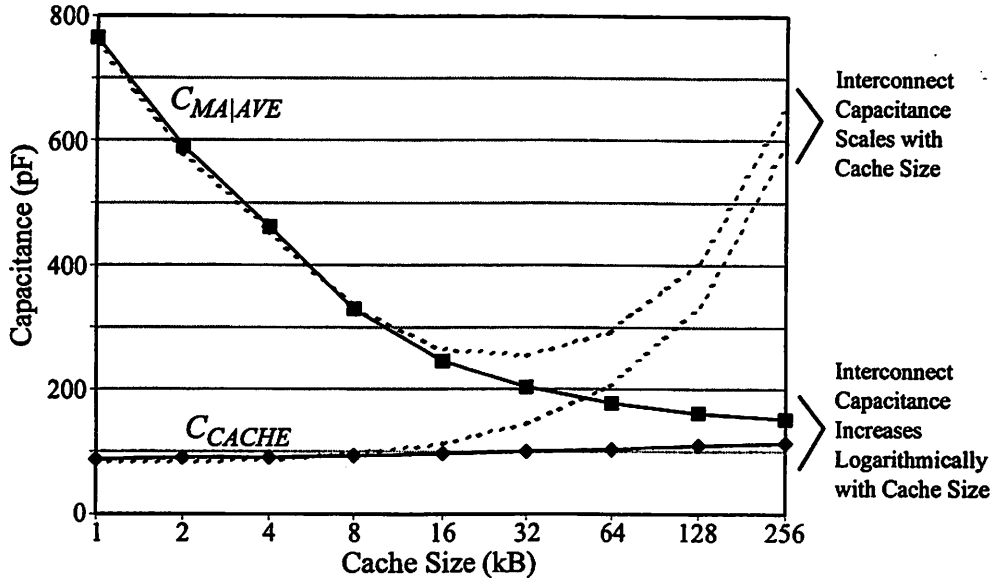


FIGURE 5.7 : Average and Cache Capacitance/access versus Cache Size (0.6µm CMOS).

prototype system, a cache size of 16kB was chosen, and was strictly limited by die-size constraints. In Figure 5.7, $C_{MA|AVE}$ was calculated based upon the cache miss rates from [henn95]. For the prototype system, benchmark simulation reported a cache-miss rate of 0.5% for a 16kB cache. This reduces $C_{MA|AVE}$ from the 250 pF/access shown in Figure 5.7 (based on a 2.0% miss rate) down to 125 pF/access for the prototype system.

5.3.2 Sub-blocking

Most large SRAMs are not composed of a single memory array, due to excessive delay on the word and bitlines, but are composed of several smaller arrays or sub-blocks in order to improve memory access time [raba96]. Enabling only the desired block, rather than the entire memory array also reduces energy consumption, as well [bund94][su95].

Reducing the size of the basic memory sub-block will reduce the energy consumed per access as well as speed up the access time. However, circuit overhead consisting of control signal generation, sense-amps, and output buffers sets the lower limit on feasible sub-block size. Figure 5.8 plots both capacitance/access and area of a 16kB cache as a function of sub-block size in our 0.6µm process. For large sub-block

sizes, the overall area asymptotically approaches a fixed value as the fractional contribution of the circuit overhead goes to zero, while the capacitance/access scales up with sub-block size. For small sub-block sizes, the capacitance/access asymptotically approaches the fixed cost due to the circuit overhead, while the overall cache size begins to exponentially increase. For sub-block sizes less than 0.125kB, the capacitance/access will actually begin to increase due to the I/O capacitance loading from the additional sub-blocks.

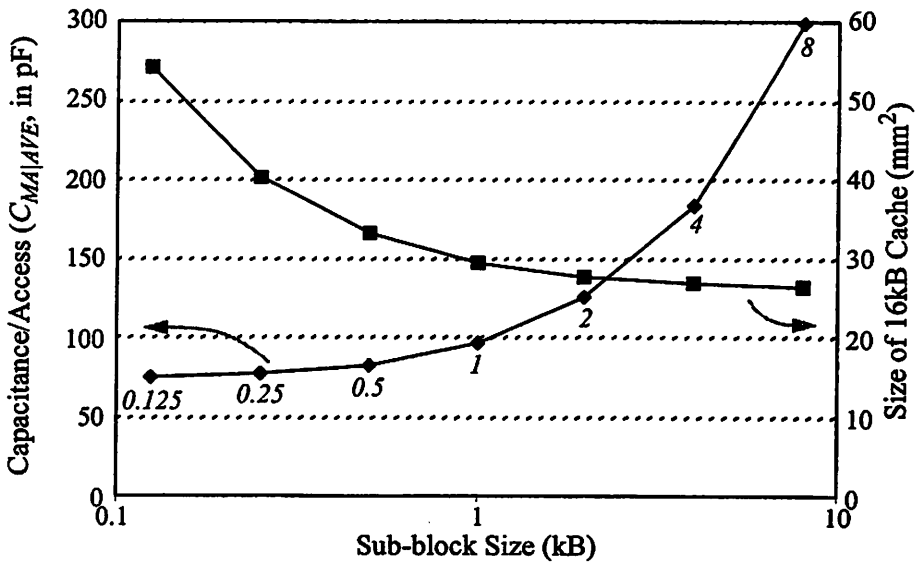


FIGURE 5.8 : Capacitance/access and Cache Size versus Sub-block Size (0.6μm CMOS).

For the prototype system, a size of 1kB was chosen, which adequately balanced the capacitance/access versus the overall cache size. At this design point, the cache contributes 30% of the total processor energy consumption, and 50% of the silicon area.

5.3.3 Tag Memory Architecture

In addition to the data memory, which hold the contents of external memory locations, an integral part of a cache is the smaller memory, called the tag memory, which is use to map cache locations to the global memory space. Multiple sequential data words are generally organized into a cache line, which is then mapped as a single cache location into the global memory space. This amortizes the cost of the tag memory

required over a larger data memory size [henn95].

The width of the tag memory is:

$$Tag\ Width = \log_2\left(\frac{Address\ Space}{Cache\ Size} \cdot Set\text{-}Associativity\right) \quad \text{(EQ 5.3)}$$

where the *Address Space* for the ARM8 architecture is 32 bits. Thus, the *Tag Width* is a minimum of 18 bits for a direct-mapped 16kB cache, and the number of bits will increase with the $\log_2$ of the set associativity. The number of tags required is:

$$Number\ of\ Tags = \frac{Cache\ Size}{Cache\ Line\ Size} \quad \text{(EQ 5.4)}$$

For the 16kB cache and 32B cache line size used in the prototype system, there are 512 tags, such that the total tag memory is on the order of 512 x 20b, or 1.28kB.

If the tags are stored in a single memory array, the energy consumed accessing a tag would be more than the energy consumed accessing the actual data word itself, since the cache is sub-blocked in 1kB arrays. However, the tag memory can similarly be sub-blocked. The same four address bits which are decoded to selectively activate the data memory sub-blocks can be used to selectively activate the tag memory sub-blocks. Then, only a 32 x 20b tag memory array is activated per cache access, significantly reducing the energy consumption per tag memory access. The increase in the memory critical path due to the gate delays added by the four-bit decoder is offset by the reduced access time of a smaller tag memory array.

5. 3. 3. 1 Design Approaches

During an access to the cache, the requested address is compared against the tag(s) where the address may potentially reside in the cache. If there is a match between the address and a tag, then the contents are currently stored in the cache, and the contents are returned to the processor core. If there is not a match, then that address location needs to be fetched from the external memory system. For an *N*-way

set-associative cache, N tags must be compared against the requested address because the requested address may reside in any of N cache line locations. There are three general approaches to implementing the tag compare:

Serial RAM Access: The tag memory is accessed first. If any of the N tags matches the requested address, then an access to the data memory is initiated. The benefit of this approach is that the data memory is only accessed for a cache hit. However, the disadvantage is that the serial RAM accesses may dominate the critical paths of the processor.

Parallel RAM Access: The tag and data memories are accessed in parallel. Thus, if any of the N tags match the requested address, the desired data word has already been read from the data memory and is available. However, writes to the data memory still must be done sequentially since the write cannot be committed to cache memory until the tag has been validated. The benefit of this approach is a fast read access, which are the dominate type of cache accesses (> 80%). The disadvantages are that cache writes must still be done serially, potentially requiring an additional cycle to complete, and extra energy is consumed for data memory reads on cache misses.

Serial CAM/RAM Access: The tag memory is implemented as a content-addressable memory (CAM). The tag CAM is first accessed, and if a match is found, then the data RAM is accessed. The benefit of this approach is that the CAM's match signals can be used directly as word line enables for the data RAM. This eliminates the need for an address decoder for the data memory, significantly speeding it up so that this approach does not dominate the critical paths of the processor. In addition, the data memory array is only accessed upon a cache hit. The disadvantage is that CAMs generally consume significantly more energy than their RAM equivalents.

5.3.3.2 Optimized Architecture

A CAM array was optimized and prototyped, which only consumes twice the

energy of an equivalently-sized, 32 x 20b SRAM memory array. This removed the key disadvantage to the Serial CAM/RAM Access approach, and therefore made it the optimal design approach to select.

By utilizing a CAM for the tag memory, 32 tags can be compared simultaneously for the same energy consumed as when comparing one tag. Thus, if the cache is 2-way set-associative, the energy consumed by the Serial CAM/RAM Access approach is the same as for the Serial RAM Access approach, because in the latter, two RAM accesses must be made to read two tags. With the CAM, the comparison between the tag and the requested address is implicit, thereby removing the need for external comparators as in the Serial RAM Access approach. Also, the Serial CAM/RAM Access approach eliminates the need for an address decoder for the data memory. Thus, for 2-way and higher set-associativity, the Serial CAM/RAM Access approach has the lowest energy/access.

With respect to access time, the Serial CAM/RAM Access approach is only slightly slower than the Parallel RAM Access approach. With the CAM, the removal of the data memory address decoder and external comparators reduces the cycle time to within 10% of the Parallel RAM approach. Therefore, for 2-way and higher set-associativity, the Serial CAM/RAM Access approach is the most energy-efficient solution for the implementation of the tag memory. Since some level of associativity was required for the prototype design, as will be explained in Section 5.3.4 below, the Serial CAM/RAM Access approach is the optimal solution.

The basic architecture is shown in Figure 5.9. The upper bits of the address, which constitute the tag, are sent to the tag CAM array. If the tag is present in the CAM, one of the match lines will go high, and indicate a cache hit. The line index bits of the address are used to demultiplex the match line to the appropriate word line of the data memory array.

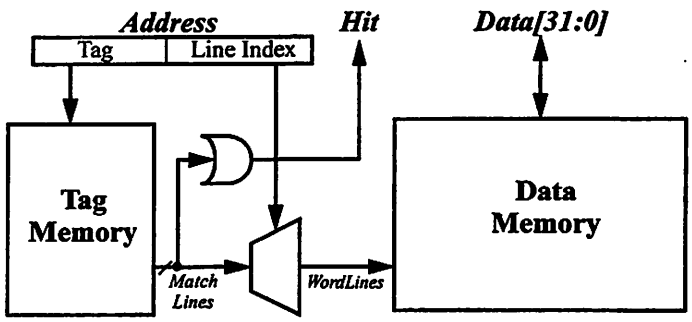


FIGURE 5.9 : Optimized CAM/RAM Cache Memory Architecture.

5.3.4 Associativity & Cache Line Size

For the ARM8 processor core used in the prototype system, the memory interface was designed and optimized for a unified cache [arm96b]. Given the constraint that the cache must be unified, a direct-mapped cache is not very desirable. When instruction and data addresses point to the same cache line, which may frequently happen, the cache will continue to alternately swap out of cache memory the conflicting address locations until the conflict is removed, needlessly spending many cycles transferring the cache lines. Thus, at least 2-way set associativity is desirable to prevent this conflict. Thus, since the processor core dictates the use of a unified cache, the cache should be designed with some level of set associativity to prevent unnecessary thrashing of the cache memory.

The requirement for at least 2-way set associativity dictated that the optimal tag memory architecture is the Serial CAM/RAM Access approach. With this architecture, the set associativity and cache line size for a fixed 1kB sub-block are related:

$$1kB = Set\text{-}Associativity \cdot Line\ Size \qquad \text{(EQ 5.5)}$$

Simulations were then used to find the optimal cache line size and set-associativity with respect to energy consumption, as shown in Figure 5.10. To measure the true impact on system energy consumption, the capacitance/access was measured

for not only the cache, which is broken down by tag and data memory, but also for the external memory system. As the line size increases, the capacitance/access contributed by the tag array decreases due to a smaller tag memory array, and because the tag memory array does not need to be accessed for sequential cache accesses to the same cache line. However, the capacitance/access contributed by the external memory increases because more words are being fetched per tag, not all of which may be used by the processor core. The contribution from the data memory array is relatively constant because it is dominated by processor core cache accesses, which is independent of cache line size, rather than cache-line reloads which are much more infrequent. Due to the low miss rate of the 16kB cache, the impact on performance is negligible for these cache-line sizes and levels of set associativity.

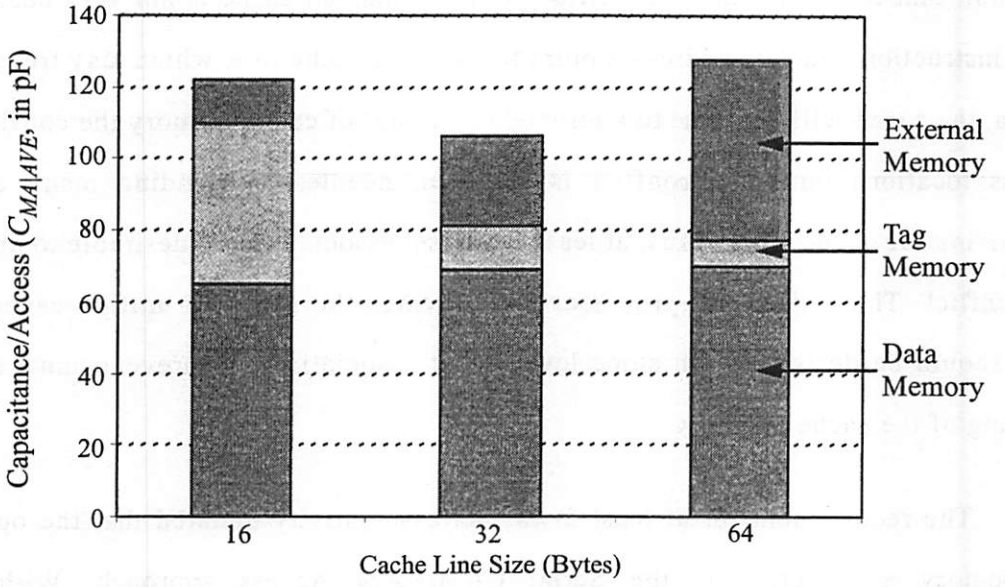


FIGURE 5.10 : Average Capacitance/Access versus Cache Line Size (0.6μm CMOS).

Thus, the optimal cache-line size for energy efficiency is 32B, which gives the cache 32-way set associativity. Simulation demonstrated that a 4-way set-associative cache would provide an insignificantly higher miss rate than a 32-way set-associative cache, but the high degree of associativity is set by a combination of the Serial CAM/ RAM Access tag memory architecture, the 1kB cache sub-block size, and the optimal cache line size of 32B.

The $C_{MA|AVE}$ reported in Figure 5.10 for the 32B line size used in the prototype system is 106 pF/access, which is below the $C_{MA|AVE}$ of 125 pF/access as reported in Section 5.3.1. This reduction of 19 pF/access occurs because this simulation better models the tag memory, which is not activated for sequential memory accesses to the same cache line.

5.3.5 Cache Policies

Simulation of the processor system running the benchmark programs was utilized to evaluate the most energy-efficient choice for the following cache policies.

5. 3. 5. 1 Write Policy

The write policy dictates what happens upon a write for a cache hit. A write-through policy dictates that every write is also transferred to the external memory to maintain continuous memory consistency between the cache and the external memory. A write-back policy only writes the data words back to main memory when a cache line that had previously been written to is removed from the cache to be replaced by another cache line.

These two policies demonstrate negligible impact on system performance, but do show a difference in the total system energy consumption as shown in Figure 5.11. For all three benchmarks, a write-through policy yielded consistently higher total system energy consumption, due to an increase in external processor bus activity. With the write-back policy, a cache line can be written to multiple times before being sent to main memory, thereby decreasing the external memory traffic. Thus, the more energy-efficient write-back policy was selected for the prototype system.

5. 3. 5. 2 Write Miss Policy

The write miss policy dictates what happens upon a write for a cache miss. A write-allocate policy will allocate space in the cache for the address being written to.

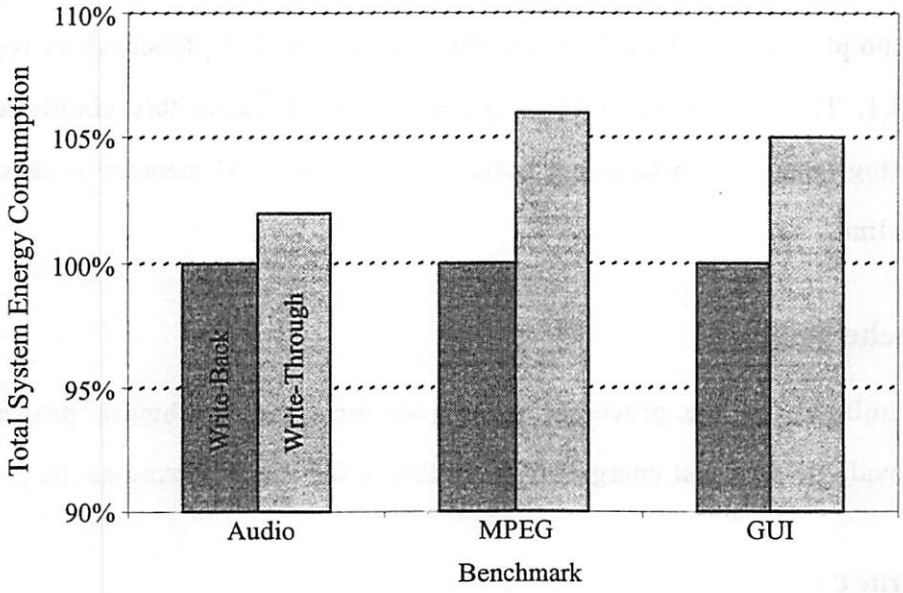


FIGURE 5.11 : System Energy Consumption vs. Write Policy for Benchmark Programs.

The cache line is first placed into the cache, and then the write can be completed. A read-allocate policy will not allocate space, but send the data word directly to main memory.

Simulations demonstrate negligible impact on both performance and energy consumption. Thus, the policy chosen was read-allocate, which simplified the design of the cache controller due to the complexity of the cache line allocation process, as described in Section 7. 2. 3. 3. External memory can support both byte and word writes, so the only added complexity was to support half-word writes, as specified by the ARM8 ISA, which are broken into two separate bytes before they are written out to main memory.

5. 3. 5. 3 Replacement Policy

The replacement policy dictates which cache line gets replaced during the cache line allocation of a read miss. There are several choices of replacement policy. One common policy is least-recently-used (LRU) replacement, which swaps out the cache line which has not been accessed for the longest time. Another policy is round-

robin replacement, which cycles through the potential cache lines. A third common policy is random replacement.

For moderately sized caches (< 64kB) and for much lower degrees of set associativity (e.g. 2-way or 4-way), the LRU replacement policy would provide the most energy-efficient policy by minimizing the cache miss rate [henn95]. However, for the high degree of set associativity (32-way) for the prototype system, simulations demonstrate negligible impact on both performance and energy consumption for all three policy variants. Thus, the simplest policy for implementation was again selected, which is round-robin. This can be implemented with 1 latch per cache line, which keeps track of the last replaced line within each cache sub-block, and advanced one cache line upon a replacement.

5. 3. 5. 4 Level-0 Cache

A level-0 (L0) cache is essentially a small buffer between the primary (L1) cache and the processor core. These are generally used when the primary cache cannot complete a memory access within one cycle, due to either a very fast processor clock speed or a very large primary cache memory [henn95]. In addition, L0 caches have been demonstrated to improve the energy efficiency of cache systems [bund94][su95]

An L0 cache contains a data and tag memory, similar to the primary cache. With an L0 cache, a memory access first checks the L0 tag memory to see if it contains the desired memory location. If so, then the L0 cache returns the desired word and the primary cache does not need to be activated. If not, once the desired memory contents have been located in the memory hierarchy, the cache line is placed into the L0 cache. If the L0 cache size is only one cache line, then the L0 can be implemented with little impact on the cache system's performance since only a single tag compare is added into the critical path. However, larger L0 cache sizes, which need to be fully-associative, require additional hardware complexity that will increase the capacitance/access of an

L0 cache hit, and may bloat the critical path and force a L0 cache miss to extend over an additional clock cycle.

With the slight modification to the CAM/RAM architecture highlighted in Figure 5.12, it can support implicit buffers, which are functionally equivalent to an L0 cache, by latching the match lines and storing the location of the previously matched tag. If a cache access has the same tag as the previous cache access, then the tag memory does not need to be enabled, and the saved match line already points to the correct word line of the data memory.

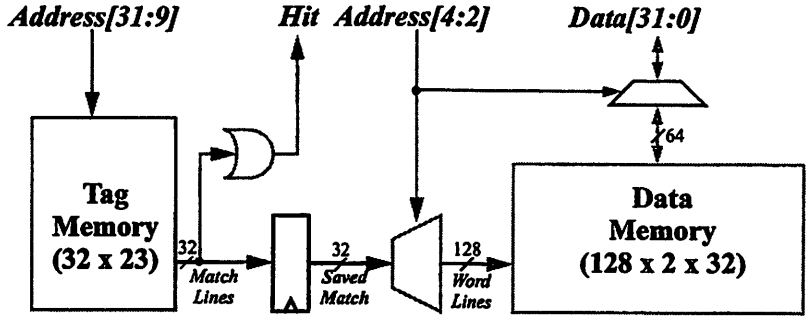


FIGURE 5.12 : Implementing an Implicit Buffer in the CAM/RAM Architecture.

While each cache 1kB sub-block has this implicit buffer, to keep all of these buffers active would require a 16-entry hash table of the last tag access to each sub-block. The complexity then becomes similar to a larger L0 cache. To simplify the hardware, only one implicit buffer can be kept active at a time, requiring the storage of only the tag of the previous cache access. This is functionally equivalent to a one cache line L0 cache.

Figure 5.13 demonstrates the improvement in energy efficiency of the L0 cache and implicit buffer by plotting the capacitance/access of a cache hit as a function of their hit rates, and comparing them to a baseline implementation which accesses the primary cache tag memory on each cache access (i.e. no L0 cache). Two sizes of L0 caches are compared, as well as the implicit buffer approach. A high-level simulation is used to estimate the capacitance/access by combining the capacitance/access of the

individual blocks with their activity factors. For low L0 hit rates, the L0 cache approach is much less energy-efficient than even the nominal case because many words are being fetched from the primary cache and placed into the L0 cache, which never get used by the processor core. Only for high hit rates does the L0 cache become more energy efficient. The implicit buffer is more energy efficient across the broadest range of hit rates, and is always more energy-efficient than the nominal case.

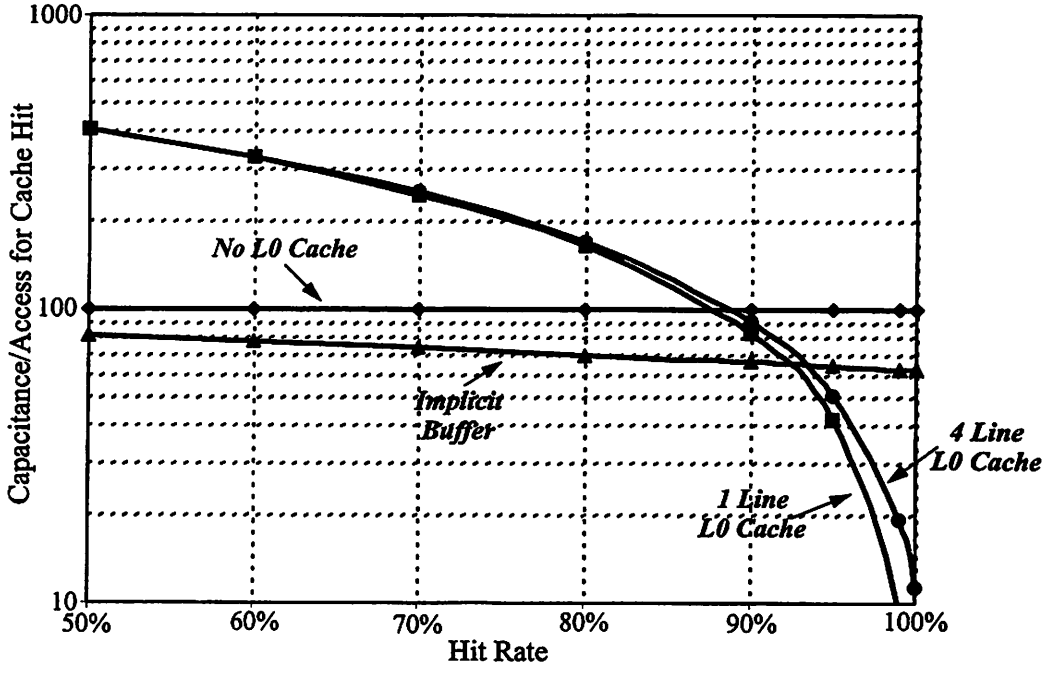


FIGURE 5.13 : Capacitance/Access of a Cache Hit for L0 Architectures (0.6μm CMOS)

For the prototype system, the implicit buffer approach was chosen. For a one-line (32B) L0 cache, the hit rate will be on the order of 80% [bund94], at which value the implicit buffer has 42% of the capacitance/access of a one-line L0 cache. A four-line L0 cache can achieve much higher hit rates, in excess of 93.5%, at which point it becomes more energy-efficient than the implicit buffer approach. However, the added design complexity of the four-line L0 cache outweighed the energy-efficiency improvement. The implicit buffer approach provides a 30% improvement in energy efficiency with minimal additional design complexity.

5.3.6 Improvement with a Write Buffer

Write stalls occur when the processor core has to be halted while it is waiting for an external write to complete. These occur for cache misses in which the cache line being discarded has been modified in the cache and needs to be updated in external memory. In addition, this occurs for direct writes to external memory which bypass the cache. These typically occur for writes to the I/O memory space, which are quite common in embedded processor systems. The use of a write buffer can eliminate these write stalls by autonomously holding the stores and cache line writes and sending them to external memory during free external bus cycles, thereby eliminating the write penalty that is otherwise present (20 core cycles per cache line, 6 core cycles per single-word write).

The basic implementation is shown in Figure 5.14. The write buffer collects external memory writes from the cache memory and processor core at the processor clock rate. When the external bus interface is free, the write buffer then completes the writes at the external bus clock rate.

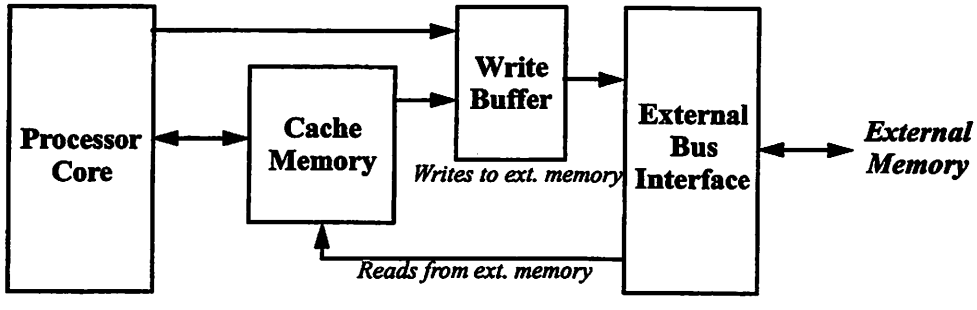


FIGURE 5.14 : Cache Architecture Utilizing a Write Buffer.

For cache line writes, the system performance improvement is minimal due to the low miss rate of the 16kB cache. The number of processor memory accesses in which a cache line needs to be written out is 0.1%, thereby reducing the processor cycles per instruction (CPI) by only 2%. However, the percentage of instructions that are writes to I/O space is 1% for the benchmark programs, which translates into a reduction of 15% in CPI, which is significant. Since the write buffer is used for only a

small fraction of the time, its energy consumption is negligible (< 0.2%). Thus the inclusion of a write buffer improves the system energy efficiency in excess of 15%.

5.3.7 Interfacing to an External Bus

The unified cache simplifies the external bus interface as compared to split instruction/data caches, in which snoop hardware is required to maintain memory coherency between the split caches [henn96]. With the addition of the write buffer, the cache system busses can be connected directly to the bus interface with no adverse affects on performance.

Bufferable writes are placed into the write buffer at the internal CPU clock rate, and while the processor continues to operate, the bus interface writes the words in the write buffer out to main memory in an autonomous fashion. For reads that must go out to main memory, the processor is halted until the desired word is available in the bus interface, but since it cannot perform useful work since out-of-order superscalar operation is not supported, no performance is lost.

5.3.8 Advantages and constraints of the ARM8 memory interface

The biggest constraint of the ARM8 memory interface is that it is designed and optimized for a unified cache, which is generally less energy-efficient than split instruction/data caches. Since a unified cache requires some level of associativity, a CAM-RAM tag memory architecture was implemented, and provided 32-way set-associativity. A unified cache has one key advantage, which is that the hardware required to maintain coherency in split caches is eliminated. The back-side of the cache can communicate directly with the bus interface and external memory.

The ARM8 memory interface contains a complex request-acknowledge handshake protocol which can be utilized to improve the performance and energy-efficiency of the cache subsystem [arm96b]. Encoded in the request control signal is

what type of request it is (load, store, instruction fetch), what size it is (byte, halfword, or word), whether the request is sequential to the last memory request, whether there are more words to follow (as part of a load/store multiple instruction), and whether the instruction fetch is speculative or not. Encoded in the acknowledge signal is whether the request completed or aborted, and how many words were successfully returned on a load/fetch request.

In addition, the cache system can send two additional signals back to the ARM8 core indicating whether the instruction and data buffers contain valid data from a previous cache load/fetch request. In the prototype system, implicit buffers were implemented inside the cache blocks themselves, as described in Section 5. 3. 5. 4. If the cache indicates to the ARM8 core that the end of the buffer has not yet been reached, then the core does not need to place the address on the internal Address bus for sequential loads/fetches, saving significant energy by not needlessly driving the bus.

5.3.9 An ARM8-optimized cache system

In addition to tuning the general properties (e.g. size, associativity) and policies (e.g. write, replacement) of the cache system to the ARM8 memory interface, further architectural design choices were implemented to take full advantage of this complex interface.

5. 3. 9. 1 Double Reads

The drawback of a unified cache is that 25% of the instructions are data transfer instructions, as measured from benchmark simulation. Thus the average number of memory word accesses is 1.25 words per instruction. A standard memory bus can only transfer one word per cycle, which would force the processor core to stall on a data transfer instruction 25% of the time. To prevent the memory bus from being a performance bottleneck, the ARM8 interface allows for two words per cycle to be retrieved from the cache system.

The data RAM in the cache is organized into two columns of 32 bits each, which is multiplexed depending upon Address[2]. By only allowing double reads to occur for even word addresses, retrieving the second word simply entails switching the multiplexer and re-firing the sense amp. Neither the CAM nor the word-line driver is reactivated to read the second word. This strategy allows two words to be returned per cycle, with the energy cost of retrieving the second word minimized to be less than 40% of a standard cache read. The penalty for this strategy is that odd-word addresses cannot return two words. However, once the prefetch unit is even-word aligned, it remains even-word aligned until the next jump to an odd-word address, so that the impact on performance is negligible.

5. 3. 9. 2 Sequential Reads

The memory interface encodes whether the address of a fetch/load request is sequential to the previous fetch/load request's address. Benchmark simulation found that only 8% of data accesses are sequential, while 70% of instruction fetches were sequential. Since only 20% of all instructions are loads, the net energy savings of an L0 cache for loads is only 0.25%. However, the energy savings of an L0 cache for instruction fetches is 10%. Thus, the implicit buffer for the L0 cache (Section 5. 3. 5. 4) was only implemented for instruction fetches, and not data loads, since only the former yields a significant reduction in overall processor energy consumption.

Using the 8-word implicit buffer simplifies the implementation of the virtual L0 cache within cache memory array. Only four bits of state are required to encode which of the sixteen 1kB blocks contains the current instruction buffer location. When the core requests a sequential instruction fetch and the address is at the beginning of a new cache line, the cache controller suppresses the CAM tag array, and the next word in the implicit buffer is read from the data memory array.

5. 3. 9. 3 Load/Store Multiple Registers

The ARM8 memory interface also encodes whether there are more loads/stores to follow sequentially, as part of a load/store multiple-register instruction (LDM/STM). If the load/store is to the cache, these operations proceed at the core clock rate, and cannot be further optimized.

However, if an STM takes a cache miss, then this encoded information allows the cache controller to packetize multiple words per address, aligned on cache-line boundaries, and place the address(es) and data words in either the write buffer, or send them directly to the external bus interface. This increases the data bandwidth on the external bus and decreases its energy consumption, compared to single-word stores, which require an address to be transmitted on the external processor bus for each data word.

If an LDM takes a cache miss, there is high probability that the missing cache line will be loaded into the cache. Thus, optimizing LDM instructions was not necessary, since the required cache lines will be loaded, and then the LDM will be serviced from the cache at the processor clock rate.

5.4 System Coprocessor

The primary role of the system coprocessor is to configure global processor settings, interface to the voltage converter chip, and maintain system control state. The coprocessor has very low performance requirements, and consumes very little energy, which for the most part is negligible. However, the energy consumption does become critical while the processor is in the sleep mode, and in part, determines the total system sleep-mode power dissipation. Thus, the coprocessor was carefully designed to minimize its standby power dissipation.

5.4.1 Architecture

The ARM8 processor core provides a dedicated coprocessor interface, through which coprocessor instructions are passed to the coprocessor unit and are executed in a parallel 3-stage pipeline. Logically, the coprocessor looks like a large register file which can be read from, and written to, by coprocessor data-transfer instructions (MCR, MRC). However, the registers themselves are very heterogeneous, and cannot be implemented as a standard register file. Some registers are read-only counters, others have hard-coded values, while others are completely virtual in that a write to them initiates some action by the coprocessor.

Thus, the coprocessor was implemented by connecting up the registers with a shared input and output bus architecture. The heterogeneous registers' bitslices were pitched-matched to provide compact layout.

5.4.2 Providing an integrated idle mode

The ARM8 core does not provide a processor halt instruction. To implement this instruction in the prototype processor, a coprocessor write instruction was used to implement this feature. Upon a write to this register, the global clock signal is halted, stopping processor operation. The processor can be restarted via an external interrupt, or an internal timer interrupt.

Since all processor state is maintained during sleep mode, the operating system can seamlessly enter and exit sleep mode without disturbing the state of the currently executing software thread.

5.5 Summary

Energy-efficient architectural optimizations at both the system level and within the cache subsystem significantly improved the overall processor system's energy

efficiency, even while limited by the fixed architecture of the ARM8 processor core. A future, energy-optimized processor core may yield even further gains in overall system energy efficiency.

In the prototype processor, the majority of the architectural optimizations occurred within the cache subsystem, and the peripheral circuitry around the processor core. Figure 5.15 plots the fraction of energy consumed by the core, and the remainder of the processor for the prototype chip, and four other implementations of the ARM architecture. The ARM8 core used in the prototype is the same as the processor core in the ARM810, and very similar to the core in the ARM940T, and SA-110. What is significantly different between these chips is the non-core component of the processor, whose energy consumption is dominated by the cache. What is demonstrated in this figure is the improvement in energy consumption of the cache subsystem, which consumes only 42% of the energy in the prototype processor. In the other processor chips, the fraction ranges from 52% to as much as 70% in the SA-110. Thus, this shows that the energy-efficient architectural design methodology presented in this chapter can provide significant reduction in energy consumption.

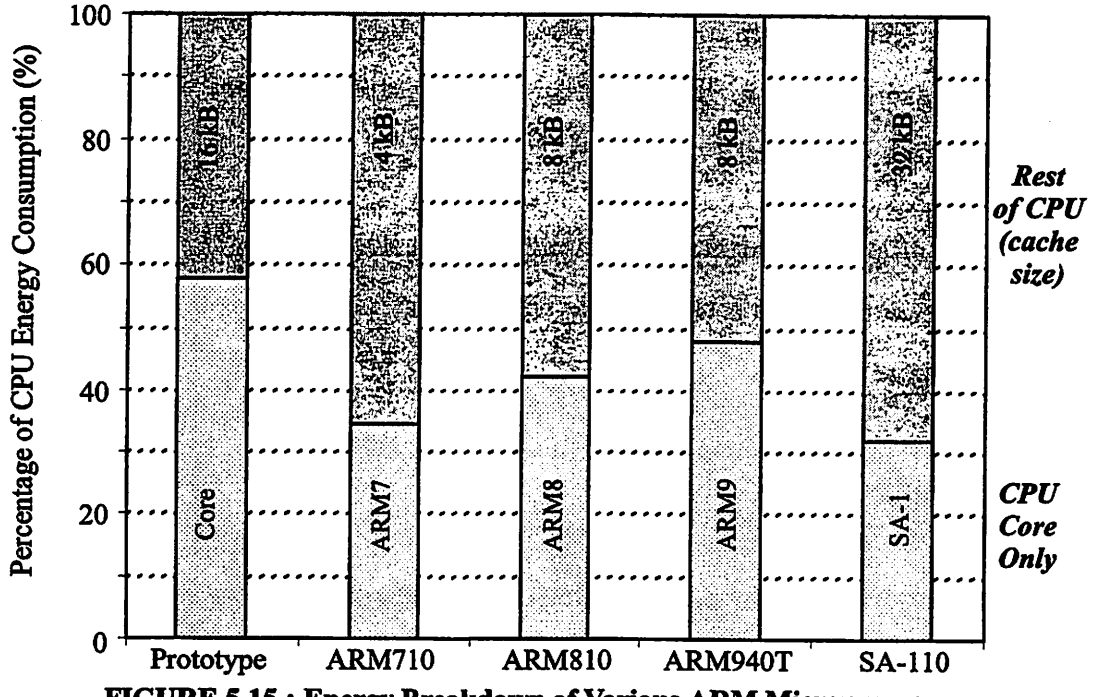


FIGURE 5.15 : Energy Breakdown of Various ARM Microprocessors.

In the case of the ARM8 processor, whose core is logically equivalent to the prototype processor, the energy of the cache subsystem has been reduced by 30% in relative terms while providing twice as large a cache. In absolute terms, the energy reduction is 61%, or more than a 2x reduction.

6 Circuit Design Methodology

The key to energy-efficient circuit implementation, much like architecture and system design, is to focus on energy consumption throughout the entire design process, rather than addressing it only as the design nears completion. There are several simple rules of thumb that will yield an energy-efficient design implementation, and to be compatible with DVS, new circuit techniques were developed for the more complex blocks such as the arithmetic and memory circuits. The last section discusses a new, ultra-low-energy bus transceiver design which was successfully demonstrated.

6.1 General Energy-Efficient Circuit Design

This section will describe a set of circuit design techniques which apply equally well to any digital CMOS integrated circuit. Many of these design techniques were first developed for low-power, custom DSP ASICs [burd94], and have been applied here to a general-purpose processor system.

In an ideal digital system, all signal paths through the circuits have equal delays, but in any practical system, this is not the case. Typical there is a small fraction of signal paths that determine the achievable cycle time, the critical paths, and in those paths, increased energy consumption may be warranted to decrease circuit delay to meet a target cycle time. All other paths should consume as little energy as possible. For

some paths, once their delay is increased to the cycle time, making them a critical path, no further energy reduction can take place. Other paths, typically with very small logic depth, have considerable slack and should be optimized solely for energy.

To make these optimizations, the circuit schematics must be analyzed and modified before being committed to layout, after which, changes become much more time intensive. Some paths reside entirely within a block, such that they can be optimized solely within the block design. Other paths cross over multiple blocks, requiring a complete schematic design of all dependent blocks for a truly optimal design, rather than simply relying on predetermined setup and hold delays based upon a behavioral model, and optimizing each block individually. The design methodology is described in much more detail in Chapter 4.

6.1.1 Logic Style

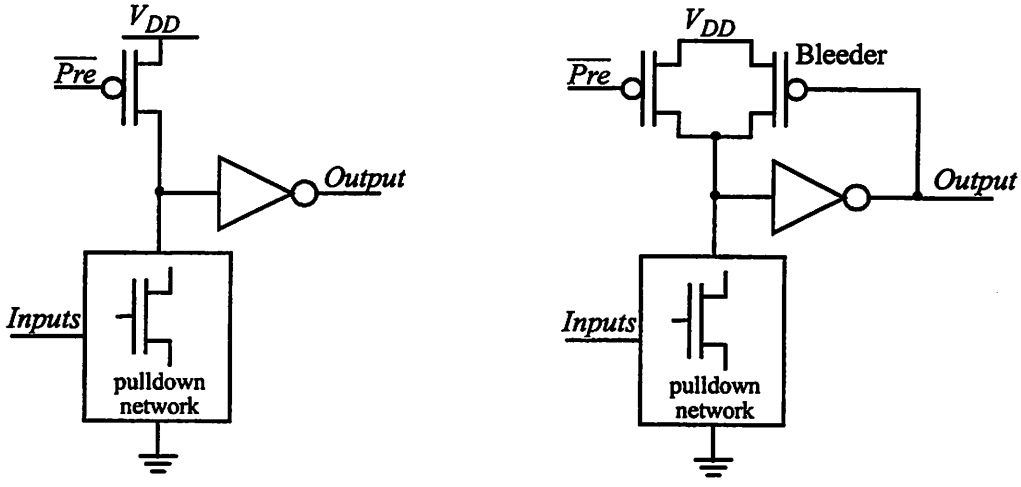
There are a variety of logic styles to choose from, such as static CMOS, CPL, Domino, NORA, C^2MOS, CVSL, etc., which vary in their delay and energy consumption [west93]. The optimal logic style cannot be found by merely selecting the one that has the smallest total capacitance. They must be compared by analyzing their effective switched capacitance per cycle, which factors in signal transition frequencies. The outputs of static CMOS and CPL only transition upon an input transition, while dynamic logic styles (Domino, NORA, etc.) incur output transitions both upon input transitions, and during the precharge phase of every clock cycle. The clock nodes in dynamic circuits have an energy-consuming transition every cycle, too. So, while dynamic logic styles tend to be faster, they often have increased energy consumption, as well.

For simple cells (e.g. AND, OR, AOI, etc.), the optimal logic style with respect to energy efficiency is generally static CMOS [burd94], while with more complex cells, there is no single, optimal logic style such that it is important to investigate a variety of

logic styles. DVS also places further restrictions on logic design.

6. 1. 1. 1 DVS Compatible Logic Design

While static CMOS is fully compatible with DVS, dynamic logic styles require some modification to ensure proper operation. Fortunately, these modifications have little impact on circuit performance and energy consumption. For buffered dynamic logic styles, which are predominantly used in the prototype design, a small bleeder PMOS device is added to maintain state on the precharged node when the inputs to the pulldown network are not actively pulling the node down, as shown in Figure 6.1. This device can have minimum width and non-minimum length, as very little current is required to maintain state.



Buffered Dynamic Logic (e.g. Domino)

Modified w/ Bleeder Device

FIGURE 6.1 : Bleeder Circuit for Dynamic Logic.

In the prototype system, all synthesized logic utilizes static CMOS logic. Dynamic logic was only selectively utilized in custom-designed blocks, such as wide-NOR gates for zero-detection operations, wide-AND gates for decoding, and shifter gates, where the effective switched capacitance reduction more than compensated for the increased switching activity.

6. 1. 1. 2 ALU Design Example

Since the delay of CMOS circuits scales well over voltage, the initial circuits were designed at 3.3V which set the target cycle time at 10ns. There were two critical data paths in the ALU, both of which had only a half-cycle, or 5ns, to complete.

One path consists of a simple shift (0, 1, 2, or 3 bits only), a selective inversion, and a 32 x 32 adder. The critical element in this path was the adder. The shifter was implemented with a four-way mux, and the selective inversion with an OR gate for a total of three gate delays. This allowed approximately 3.5ns for the addition to complete, taking into account latch setup and hold time requirements. Various adders were analyzed, including ripple, carry-select, and Brent-Kung adders [west93]. The latter was selected because it could meet the targeted delay in the minimal energy consumption. Other adders, such as the ripple-carry, had lower energy consumption, but were removed from consideration because they could not meet the delay target.

The other path consists of a fully-programmable 32-bit shift, and a logic operation unit. The logic operation unit (AND, OR, XOR) maps to a single combinational logic gate, with an additional gate required for buffering. The shifter had approximately 3.5ns to complete its operation, as well. The natural implementation of the shifter would be a barrel shifter, which is the most compact. However, for DVS compatibility, the usual NMOS pass gates must be replaced with CMOS pass gates. This causes the 32-bit barrel shifter to consume a large amount of energy due to the large CMOS pass gates required to keep delay through the shifter minimized. Instead, a logarithmic shifter was utilized to reduce energy consumption, and was tuned to meet the target cycle time.

6.1.2 Transistor Size

Traditional design methodologies utilize cell libraries with transistor sizes larger than necessary. A typical "1x" output driver size uses transistor widths much

larger than the minimum size. This is due to the desire to provide maximum drive capability under a wide variety of load conditions. While this increases gate-area density and simplifies the cell libraries required, it is not energy-efficient.

A more energy-efficient solution is to set the base "1x" driver size to be minimum size. For our MOSIS 0.6μm process, the minimum NMOS width is 1.2μm. To equalize rise and fall times, and minimize gate delay, the "1x" PMOS width is 2.4μm due to the lower mobility of PMOS devices. A simple, energy-efficient, transistor-sizing methodology is to initially set the size of all transistors so that short-circuit current is minimized, as will be described in Section 6. 1. 2. 1, below. Transistors in the critical paths are then increased in size to decrease delay, as necessary, and all remaining transistors with small fan-out are reduced in size while not violating constraints for minimizing short-circuit current.

Conventional belief is that as process technology improves, interconnect capacitance will dominate the total capacitance on a node, making transistor-size dependent capacitance (gate oxide and diffusion capacitance) insignificant. Thus, the optimal transistor size is much larger than minimum size, since performance will increase while having a negligible impact on energy consumption. However, this is not true, since while interconnect capacitance will dominate the global nets, for local nets, transistor parasitic capacitance will continue to be significant, and remain critical to minimize whenever possible [sylv98][ho99].

6. 1. 2. 1 Minimizing Short-Circuit Current

By bounding the ratio of input to output rise/fall times between gates, short-circuit current energy consumption can be minimized. If the ratio is kept to less than two, the upper limit of additional energy consumption is 12% at $V_{DD} = 3.3V$. This is achieved by sizing up devices as necessary when driving large loads. This constraint will be defined as:

$$MIORFT = Maximum\ Input\text{-}to\text{-}Output\ Rise/Fall\ Time = 2 \qquad \text{(EQ 6.1)}$$

The simplest gate construct consists of minimum-size, back-to-back inverters. In our 0.6μm technology, the minimum nodal capacitance between these gates is 13.5fF (50% gate capacitance, 50% diffusion capacitance). With a minimum interconnect capacitance of 1.5fF, the minimum nodal capacitance rises to 15fF.

In Figure 6.2, a histogram of the nodal capacitance of the prototype ARM8 core is shown out to 50fF. The first peak occurs due to the small diffusion capacitance between the numerous series transistors. These nodes represent internal gate nodes and are not relevant. The next peak, starting around 15fF, represents inter-gate nodes and validates the previous estimate. Nodes in the 10-15fF range occur either for other internal gate nodes, or when the PMOS size has been reduced below the "1x" width of 2.4μm, as will be discussed later.

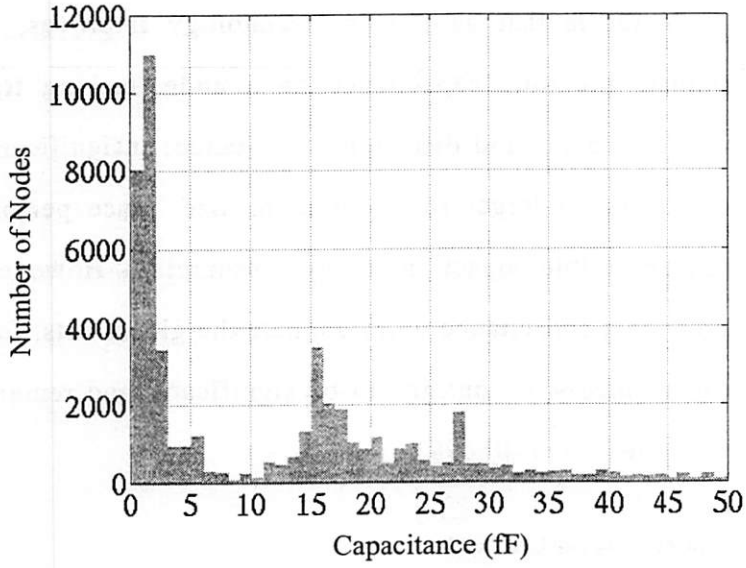


FIGURE 6.2 : Histogram of Nodal Capacitance for ARM8 Core. (53k nodes)

The minimum load capacitance driven by a "1x" gate is 15fF. The test circuit in Figure 6.3 was used to find the maximum load capacitance that a "1x" gate could drive while meeting the MIORFT. The worst case occurs when a driver with a maximum load capacitance drives a gate with minimum load capacitance.

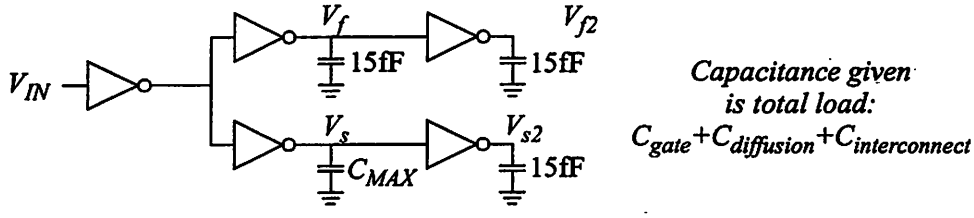


FIGURE 6.3 : Test Circuit for Finding C_{MAX} for a 1x Gate Output Driver.

SPICE simulation yielded a C_{MAX} of 50fF. This results in a rise/fall ratio for V_s/V_f of 2.85. But what is critical is the input-to-output rise/fall ratio, and since the longer rise/fall time on V_s degrades the rise/fall time on V_{s2} the rise/fall ratio of V_s/V_{s2} is 1.9, which is below the MIORFT of two.

For larger load capacitances, the driver transistor sizes are just scaled up proportionally. A "2x" gate can drive 50-100fF while meeting the MIORFT constraint; a "3x" driver can drive 100-150fF. While a "3x" driver could drive 45-150fF and still meet the MIORFT, the finer resolution on the bins helps to further minimize energy consumption. By using a "2x" instead of a "3x" to drive a 100fF load, the combined capacitance of transistor parasitics and output load has been reduced 10%.

6. 1. 2. 2 Critical Paths

The timing verification methodology in Section 4.6 is used to identify paths that exceed the target cycle time. Within these paths, gate sizing can be increased to reduce the path delay. The gate delay for the "1x" driver varies by 40% over the range of rated load capacitance. Gates can be sized up to significantly reduce delay at the expense of increased energy consumption. Once the target cycle time has been met, with some headroom, further size increases are not necessary. Since the number of paths that are critical and need to be resized are small, the overall increase in chip energy consumption is insignificant.

To prevent paths from arising that cannot be resized to meet the target cycle time, a maximum logic depth constraint is imposed on the schematic design. This logic

depth can be calculated by finding the maximum number of minimum sized inverters in series in which the delay through them is below the target cycle time by some headroom margin. For a 10ns (at 3.3V) target cycle time, and including 10% headroom margin, the maximum logic depth is 30 gates (single inversion, e.g. NAND, NOR, AOI, etc.) per half-cycle.

Thus, a schematic can be guaranteed by design that its layout implementation can meet the target cycle time, preventing radical circuit redesign. If a netlisted schematic has paths with logic depths greater than 30, then the circuit must be redesigned through logic compaction or architectural modification to reduce the logic depth to the allowed amount.

6.1.2.3 Non-Critical Paths

Paths that have very little logic depth can be made as slow as reasonably possible without impacting target cycle time. However, to minimize short-circuit current, gates with drive strength larger than "1x" are not candidates for size reduction.

Simple gates within a more complex cell, such as an adder or flip-flop, often have minimal capacitive loading. Hence, the PMOS width can be reduced from the nominal 2.4μm down to 1.2μm without exceeding the MIORFT while decreasing the gate-oxide and diffusion capacitance by roughly 33%.

Standard cells are not good candidates for size reduction because their output loading is not known until after place & route, and can change with subsequent re-routes. Custom datapath cells, however, make excellent candidates because the internal loading is known at the time of cell creation. Thus, down-sizing of transistors is done only within custom datapath cells.

6.1.3 Gated Clocks

Gating, or selectively enabling, clocks is critical for energy-efficient circuit

implementations, and this is particularly true for general-purpose microprocessors, in which the clocked elements typically require activation only a fraction of the time. The clock drivers described in Section 4. 3. 2. 1 contain inputs for both a local and global clock enable signal, which allows either entire sections of the processor (e.g. processor core, cache, etc.) to be halted with a global signal, or fine-grained control with a local signal.

The local enable signal is used whenever the necessary condition for clocking a latch can be calculated from locally available control signals. Routing additional wires across the processor to provide the necessary state information, and adding a large amount of additional logic to calculate the local enable signal can be less energy efficient than always clocking the latch while the global enable signal is asserted. Thus, for latches where the necessary state information is not readily available, the energy penalty for providing this information must be evaluated and compared with the energy savings of having a local clock enable signal.

In the prototype system, a total of 256 clock drivers were distributed across the chip as shown in Figure 6.4. Of these, 80% have a local enable signal, demonstrating that fine-grained clock gating can be utilized quite extensively in the processor design. Within the processor core itself, 60% of the drivers are locally enabled.

These clock drivers, in turn, drive a total of 6292 latches distributed across the processor chip as shown in Figure 6.5. This number does not include memory elements in the register files and in the cache memory. If these latches were clocked every cycle the processor is active, the aggregate clock load would be 150 pF/cycle, which is one-half of the entire processors capacitance/cycle while it is active. However, 75% of these latches were driven with a locally-enabled clock driver, reducing the average, aggregate clock load to somewhere in the 50-75 pF/cycle range. Unfortunately, an exact number is difficult to quantify.

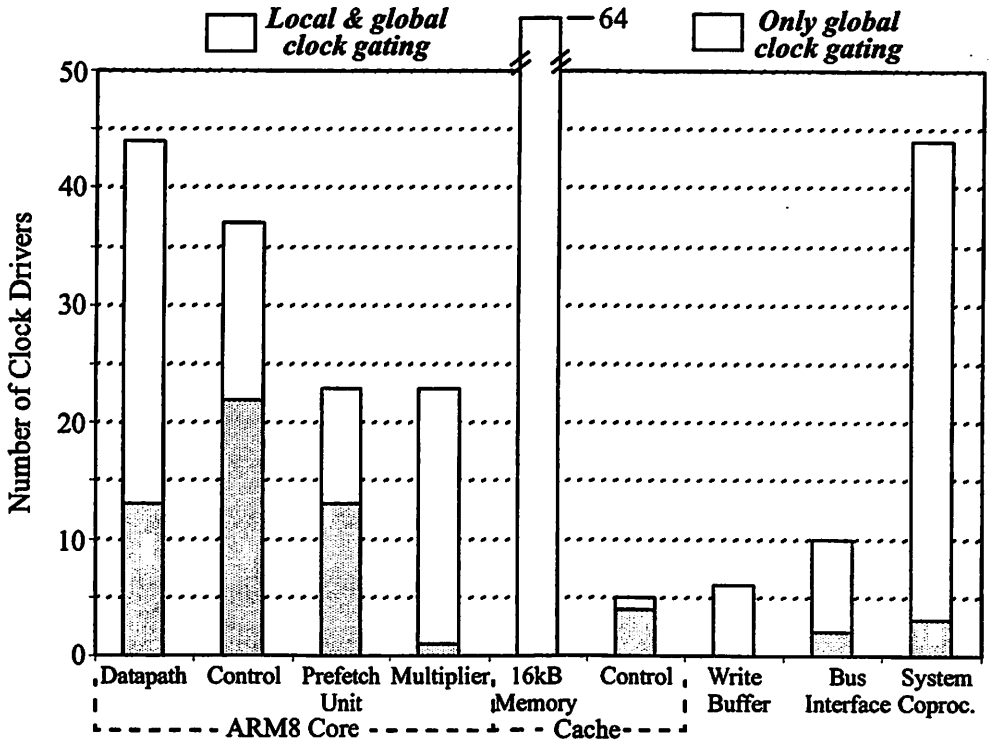


FIGURE 6.4 : Clock Drivers in the Prototype Microprocessor

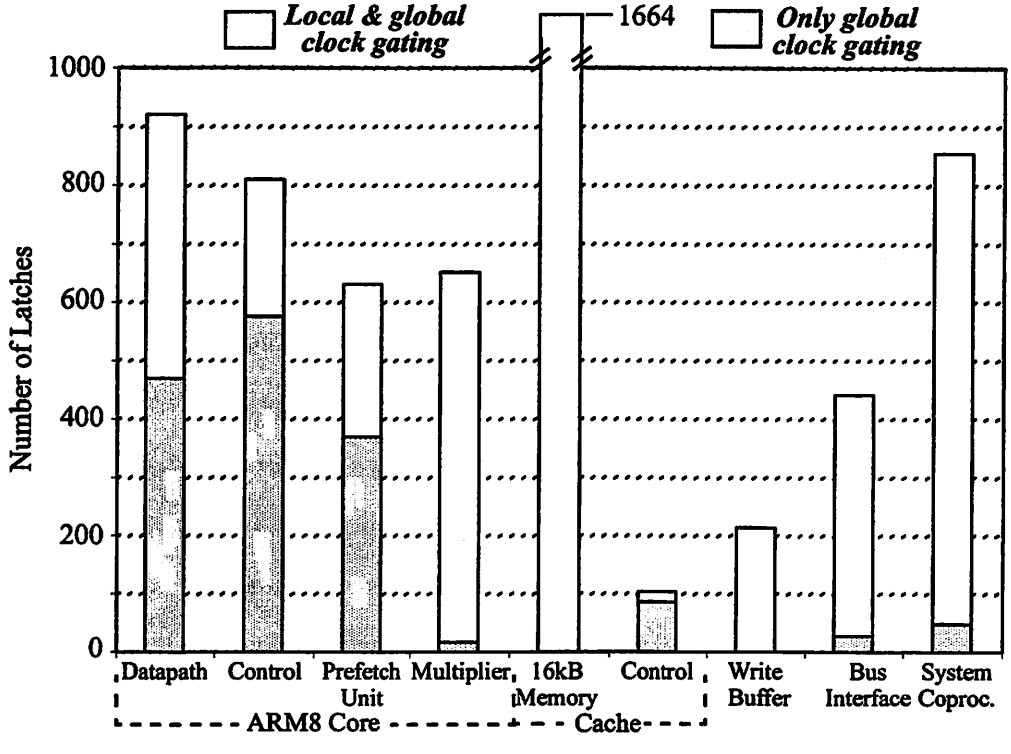


FIGURE 6.5 : Latches in the Prototype Microprocessor

Finally, some latches must be clocked every cycle, even while the processor is completely halted, and they are required in any block that interfaces with the external world (e.g. interrupt controller, memory controller, etc.). Since these latches contribute to the idle power dissipation, it is important to keep the number to a bare minimum. In the prototype design, 60 latches required latching every cycle, and contributed 10μW to the idle power dissipation, which is on the same order of magnitude as the subthreshold leakage current power dissipation.

6.1.4 Optimizing Interconnect

The metal profile for our 0.6μm process is shown in Figure 6.6, along with the capacitive components of a representative, minimum-width *Metal2* wire. In this process technology, minimum-width wires have almost a square profile, and as process technology continues to advance, the height of the wires will become significantly greater than their width.

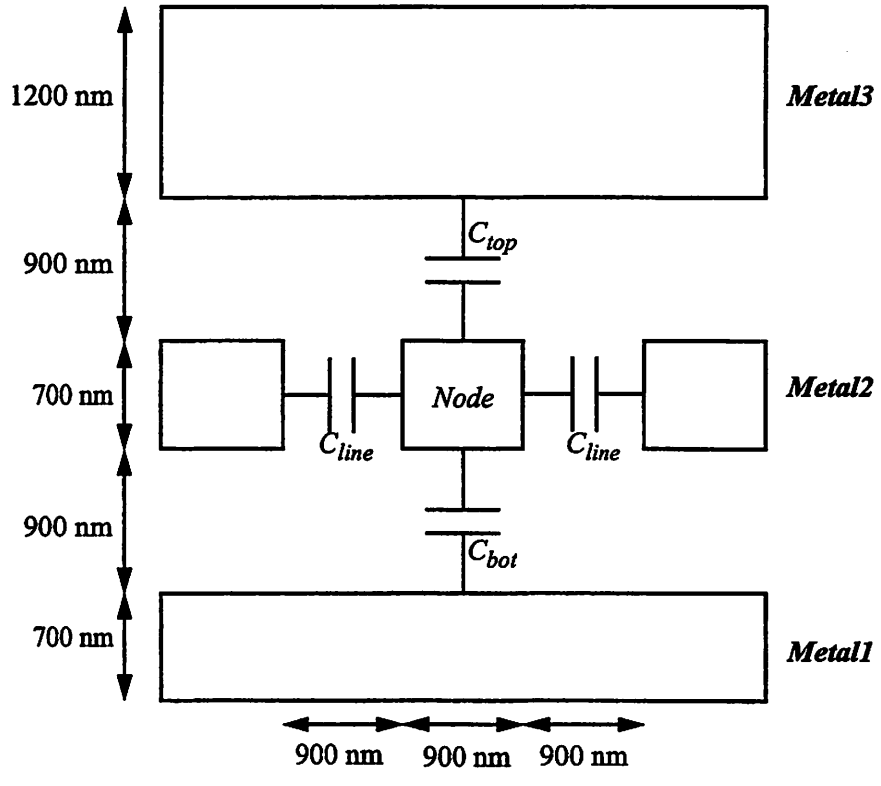


FIGURE 6.6 : Interconnect Dimensions and Capacitance Components (MOSIS 0.6μm)

The total capacitance on *Node* is:

$$C_{TOTAL} = C_{top} + C_{bot} + 2 \cdot C_{line} \qquad \text{(EQ 6.2)}$$

where the line capacitance, C_{line}, accounts for only 11% of C_{TOTAL} with *Metal1* and *Metal3* present, but 43% in the absence of *Metal1* and *Metal3* as shown in Figure 6.7. In areas of the chip with dense signal routing on all layers, spacing *Metal2* wires at twice-minimum spacing can reduce line capacitance by 11%, or more, depending upon how much *Metal1* and *Metal3* is present around the wire. In regions of the chip loosely populated with wire routes, spacing wires far apart can provide a significant reduction of almost 2x in energy consumption. This is particularly true for *Metal2* and *Metal3* wires, as they are farther from the substrate than *Metal1*.

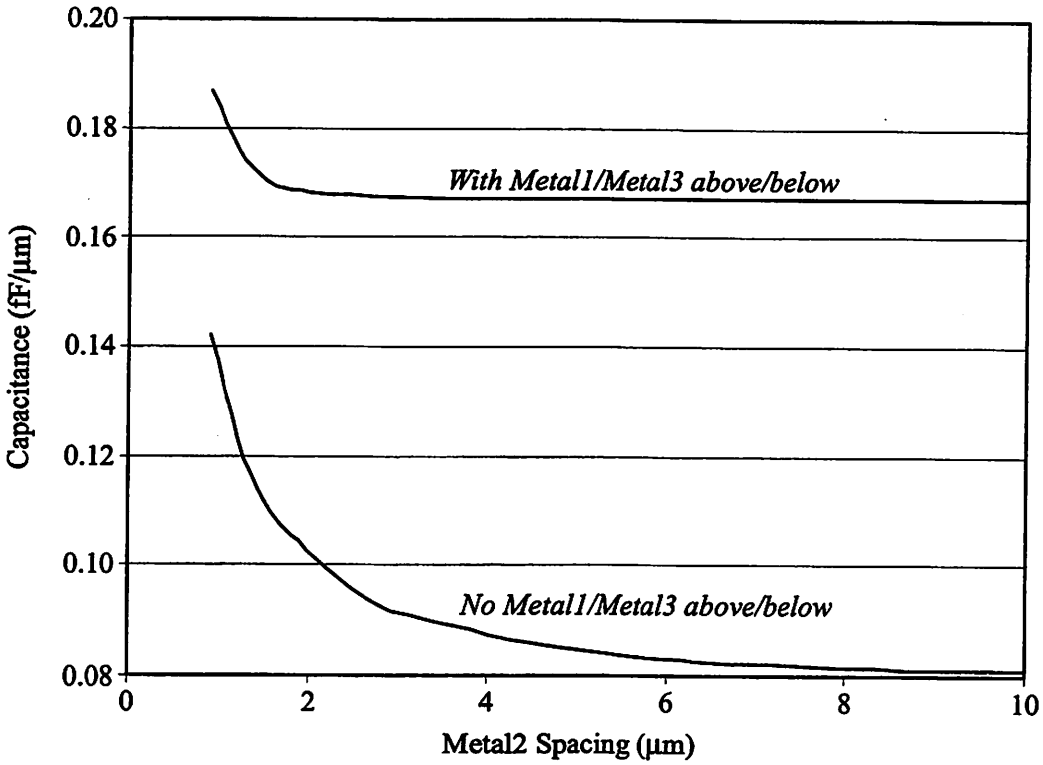


FIGURE 6.7 : *Metal2* Wire Capacitance/µm With Adjacent *Metal2* Wires.

In the prototype microprocessor chip, this technique was used pervasively to reduce energy consumption. *Metal3* feedthru wires over the datapath were spaced equidistantly to minimize their overall capacitance. The channel routes in the ARM8

core and in the cache system, in sparsely populated regions, were also spaced farther apart. To minimize the load on the global clock, which transitions every single cycle and is by far the highest energy-consuming net, it was routed in *Metal3* with at least 10μm to the nearest *Metal3* wire. *Metal2* and *Metal1* wires were only utilized to cross underneath and perpendicular to the clock net.

In more advanced process technologies, the fraction of C_{LINE} to C_{TOTAL} goes up, which just exacerbates the benefit of spacing wires farther apart than minimum spacing. Copper wires reduce the height of the metal wires, but this height reduction is much less than the lateral geometry shrink going from our 0.6μm process to a much more advance 0.18μm copper process technology.

6.1.5 Layout Considerations

Many layout optimizations that are done for performance improvement or silicon-area efficiency also improve circuit energy efficiency. For example, the layout constraints of the custom datapath cells and the standard cells were carefully optimized to minimize the silicon area in our 3-metal 0.6μm process technology, and by doing so, the overall circuit energy efficiency was increased.

Fingering devices can be used to reduce drain capacitance to not only speed up circuit performance, but reduce the energy consumption. To further reduce drain capacitance, pass gate diffusion can be merged with a driver's diffusion region. Spacing control signals that frequently transition, such as clock signals, away from other cell geometries reduces energy consumption as well.

6. 1. 5. 1 Datapath Cell Layout

The datapath cell pitch was not set until the entire schematic of the ARM8 core datapath was complete, so that the absolute minimum number of cell feedthrus could be calculated. The initial design yielded thirteen feedthrus, which after schematic

redesign, was reduced to ten feedthrus. In addition, further constraints had to be specified to optimize the layout with only three metal layers available for routing.

Within the cell, as shown in Figure 6.8, vertical *Metal1* wires were used to route power and ground. They have higher resistance than *Metal3* wires, but require many less contacts to connect to the devices, and also maximizes capacitance on the power/ground lines. Their width was dictated by the maximum current the cell could draw (Section 4. 4. 2. 2). *Metal2* was utilized for control lines that span the entire datapath, as well as local cross-overs of the *Metal1* power/ground lines. They were spaced as far apart as possible to reduce parasitic capacitance on them. *Metal3* was used exclusively for feedthrus. The minimum pitch of *Metal3* to accommodate contacts is 2.55μm, so in order to allocate room for ten feedthrus, the cell height was set to 25.5μm. These constraints minimized the overall area of the datapath, which in turn minimized the length of the long feedthrus across the datapath, and reduced their capacitance and energy consumption, as well.

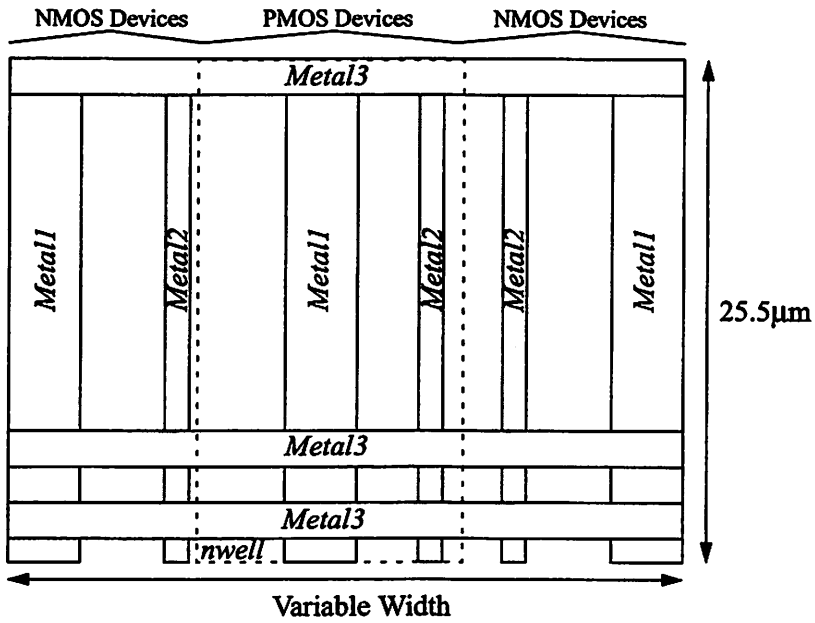


FIGURE 6.8 : Datapath Cell Layout Constraints.

The cells were designed so that they can abut on top and bottom. which allowed them to be tiled up to form the datapath. By placing the power/ground lines on

the far left and right of the cell, it can also directly abut other datapath cells on either side if a routing channel is not required. In better process technologies with more metal layers available, the cells can always abut because the additional metal layers remove the need for explicit routing channels.

6. 1. 5. 2 Standard Cell Layout

In designing the standard cell library, the goal was to minimize the overall area of synthesized layout, which was achieved by reducing the cell size as small as possible. The pitch was set to 19.2μm, as shown in Figure 6.9, which allowed for a twice-minimum size PMOS device to be placed without having to finger it. To free up as much *Metal2* as possible for the router, no *Metal2* was allowed inside the cell for routing, and all pins had to be placed, centered about the middle in *Metal2*, on a 2.4μm routing pitch. This allowed the router to use *Metal2* over the cell. The router used *Metal3* horizontally over the cell, *Metal1* horizontally outside the cell, and *Metal2* for vertical routes. The cells had to be designed to abut on either side.

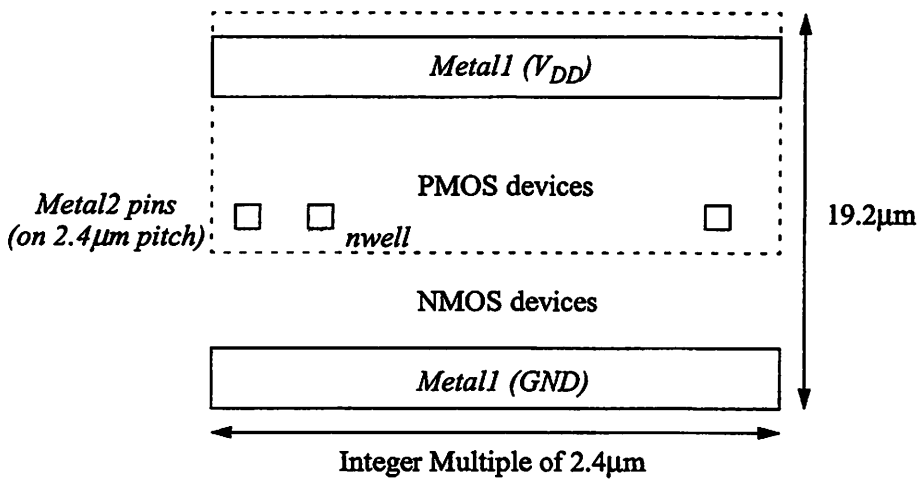


FIGURE 6.9 : Standard Cell Layout Constraints.

6.2 Memory Design

The basic memory blocks used in both the cache memory and the external SRAM have been derived from a previous design which utilized sub-blocking, self-

timing, and charge-sharing sense-amplifiers for a very low-energy implementation [burs97]. In addition, this design was extended to a CAM which was utilized for the cache tags. The key modification made to the previous design to make it DVS compatible was changing the charge-sharing sense-amplifier, which uses an NMOS pass gate to limit the signal swing on the bitlines to V_{DD} - V_T, to a full-swing design.

The critical aspect of a memory design, in order to ensure DVS compatibility, are those circuits which are not standard CMOS logic. These primarily include the memory cell, which only pulls down the bitline voltage by some fraction of V_{DD}, and the sense-amp circuit which restores the signal on the bitlines to full-scale. While allowing the bitlines to swing full-scale would improve circuit robustness for DVS, this would significantly increase memory energy consumption and delay, and is therefore not a viable option.

The rest of the memory circuits (i.e. address decoder, word-line driver, output buffer, and control circuitry) are typically implemented with standard static or dynamic CMOS logic, for which the circuit delay scales with voltage similarly to any other logic circuits.

6.2.1 SRAM

The critical part of the SRAM's signal path along the bitlines, including the memory cell and sense-amplifier circuits, is shown in Figure 6.10. The width of the sense amplifier layout is twice that of the memory cell, so a 2-to-1 multiplexer is used for column decoding to provide efficient, compact layout by pitch-matching the sense amplifier to two memory cells. CMOS pass gates are required to implement the bi-directional multiplexer.

The bitlines on either side of the multiplexer are precharged to V_{DD}, so while the SRAM is not being actively accessed, these precharged nodes will vary in voltage with V_{DD}. The internal state of each memory cell, which is maintained by the cross-

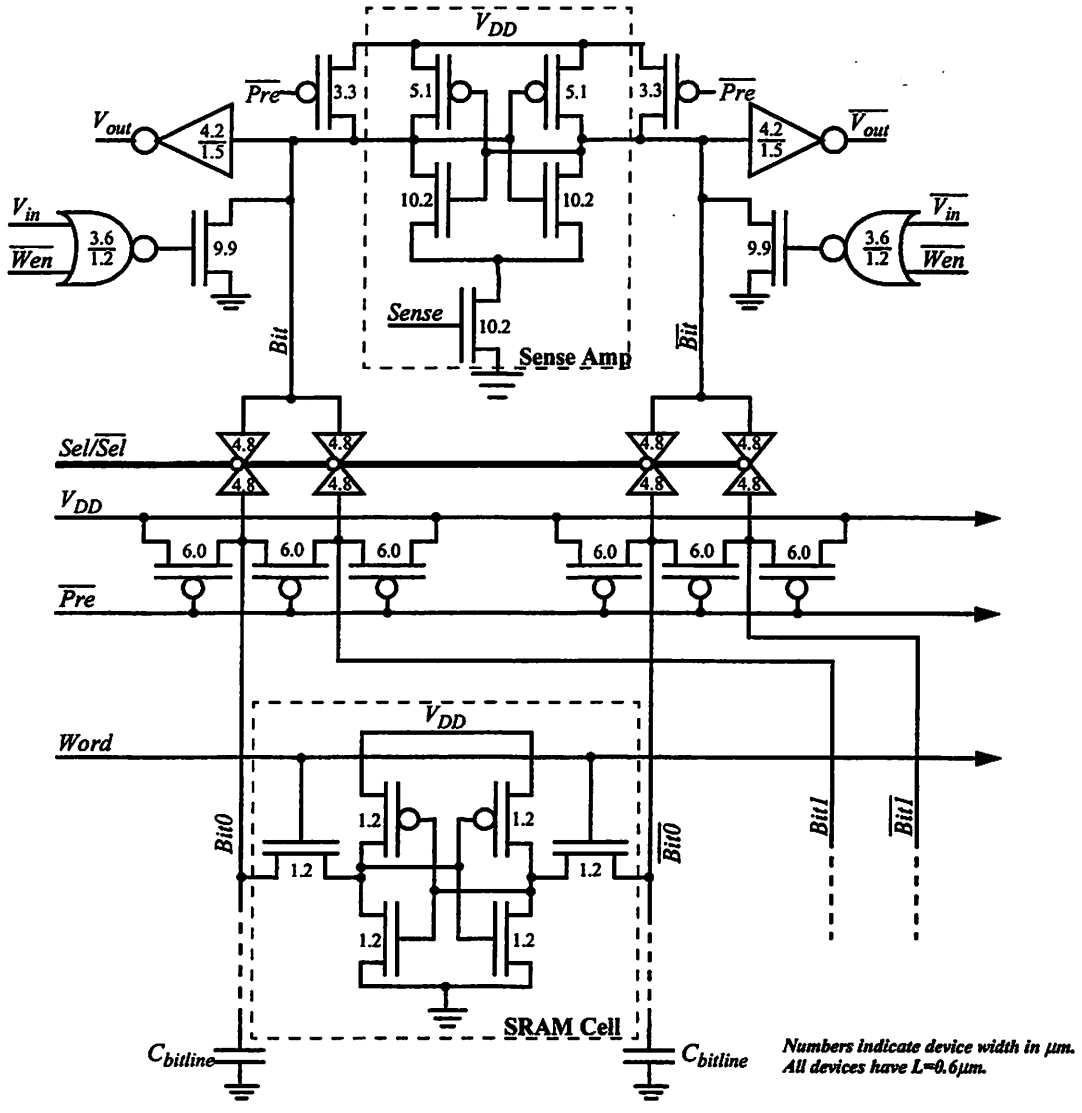


FIGURE 6.10 : SRAM Cell and Sense Amp.

coupled inverters, will also scale in voltage as V_{DD} varies. Thus, when the SRAM is inactive, it can tolerate transient variations on V_{DD} much like static CMOS circuits because all logic high nodes are actively being pulled up by a PMOS device.

Writing to the SRAM cell requires pulling one of the bitlines all the way to ground in order to flip the state of the cell's cross-coupled inverters. This is accomplished by one of the NMOS pull-down devices on *Bit* and $\overline{Bit}$, which is enabled by its corresponding NOR gate when the write enable (*Wen*) signal is high. The delay of this signal path will scale with varying V_{DD} much like static CMOS logic.

When the SRAM cell is being read from, the cross-coupled inverter, whose output is low, begins to pull down one of the bitlines through the NMOS pass-gate activated by the *Word* signal. Both NMOS devices are minimum size to reduce the size of the SRAM cell (which determines the total SRAM block size) and therefore can only pull the bitline down slowly. The sense amplifier is used so that only some fraction of the voltage V_{DD} has to be developed across the bitlines to register a signal transition.

The memory is self-timed, which in addition to minimizing switching activity to significantly reduce energy consumption, also enables the delay of the SRAM read to scale with varying V_{DD} similar to static CMOS logic. A dummy word line is used to generate the *Sense* signal, which is delayed from the activation of the *Word* signal by:

$$t_{Word \rightarrow Sense} = \frac{2 \cdot C_L \cdot V_{DD}}{I_1(V_{DD})} \qquad \text{(EQ 6.3)}$$

which is just the delay through two static CMOS gates. The voltage differential generated on the bitline at the input of the sense-amp when it is activated is:

$$\Delta V_{Bit|0} = \frac{I_2(V_{DD}) \cdot t_{Word \rightarrow Sense}}{C_{Bitline}} = \frac{I_2(V_{DD}) \cdot C_L \cdot V_{DD}}{I_1(V_{DD}) \cdot C_{Bitline}} = \alpha \cdot V_{DD} \qquad \text{(EQ 6.4)}$$

where α is a voltage-independent term because the voltage-dependence in the ratio of the current terms, I_1 and I_2, cancels out in Equation 6.4. Thus, the voltage drop is proportional to V_{DD}. The delay from the activation of the *Sense* signal, at which point the voltage on *Bit* is $\Delta V_{Bit|0}$, to the *Bit* signal crossing $V_{DD}/2$ so that a signal transition registers on V_{out}, is:

$$t_{Sense \rightarrow Bit} = \frac{C_{Bit} \cdot \Delta V_{Bit}}{I_{sa}(V_{DD})} = \frac{C_{Bit} \cdot \left((V_{DD} - \Delta V_{Bit|0}) - \frac{V_{DD}}{2} \right)}{I_{sa}(V_{DD})} = \frac{C_{Bit} \cdot V_{DD} \cdot (0.5 - \alpha)}{I_{sa}(V_{DD})} \qquad \text{(EQ 6.5)}$$

where I_{sa} is the average current in the sense-amp's series NMOS transistors as *Bit* varies from V_{DD} - $\Delta V_{Bit|0}$ to $V_{DD}/2$, and scales with V_{DD} similar to a static CMOS gate.

Thus, with self-timing, the voltage differential generated at the input of the sense-amp is proportional to V_{DD}, which then allows the delay of the signal path from

Word to V_{out} to scale with V_{DD} similar to static CMOS logic. This is demonstrated in Figure 6.11 which plots this delay versus static CMOS logic over V_{DD}.

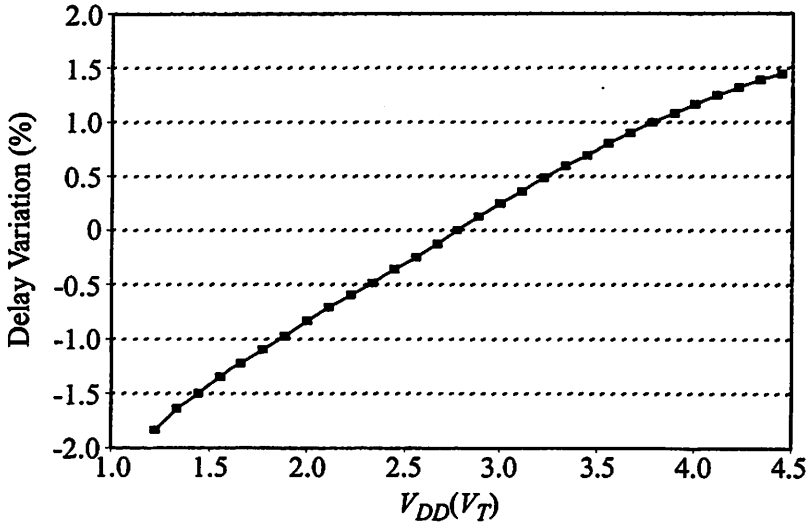


FIGURE 6.11 : Relative Delay from *Word* to V_{out} vs. Static CMOS logic for Constant V_{DD}.

So while this delay tracks well for constant V_{DD}, when V_{DD} dynamically varies while the sense-amp is evaluating, this delay begins to deviate. This occurs because the voltage on *Bit* remains independent of V_{DD} during the sensing, so that while the voltage differential on *Bit* to flip the sense-amp at constant voltage is proportional to V_{DD}:

$$\Delta V_{Bit} = (V_{DD} - \Delta V_{Bit|0}) - V_{DD}/2 \qquad \text{(EQ 6.6)}$$

when V_{DD} varies by ΔV_{DD}, the required voltage differential scales inversely with ΔV_{DD}:

$$\Delta V_{Bit}(\Delta V_{DD}) = (V_{DD} - \Delta V_{Bit|0}) - (V_{DD} - \Delta V_{DD})/2 = \Delta V_{Bit} + (\Delta V_{DD})/2 \qquad \text{(EQ 6.7)}$$

Thus, when ΔV_{DD} is positive, indicating that V_{DD} is falling, the amount of voltage required to switch, $\Delta V_{Bit}(\Delta V_{DD})$, actually increases with V_{DD} and causes the sense-amp to slow down much faster than static CMOS logic. Likewise, when ΔV_{DD} is negative, indicating that V_{DD} is rising, the value $\Delta V_{Bit}(\Delta V_{DD})$ actually decreases with V_{DD}, and causes the sense-amp to speed up much faster than static CMOS logic. As shown in Section 3.4.3, this issue is most critical at low V_{DD}, and ultimately limits how fast V_{DD} can be allowed to vary. This is a fundamental limitation of sense-amps.

6.2.2 CAM

A traditional implementation of a CAM cell is shown in Figure 6.12, which uses a dynamic NOR gate (*M3*) to generate a *Match* signal that remains high if the data values placed on the *Bit*/$\overline{Bit}$ lines completely matches the cells' contents across the entire row [west93]. If any one bit in the row mismatches against the input pattern (i.e. $Bit \neq m$ and $\overline{Bit} \neq \overline{m}$), one of the NMOS pass gates, *M1* or *M2*, pulls the input to *M3* high, which in turn pulls *Match* low.

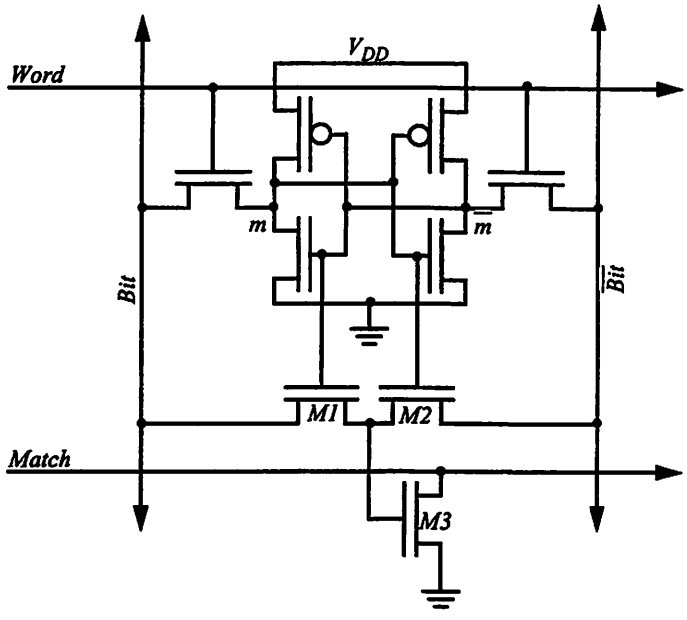


FIGURE 6.12 : Traditional CAM Cell.

This implementation is not DVS compatible, due to the NMOS pass gates. At least two more PMOS transistors are required to convert them to CMOS pass gates. Also, the traditional CAM cell requires *Bit* and $\overline{Bit}$ to be pre-discharged for a match operation, and is not compatible with read and write operations which require *Bit* and $\overline{Bit}$ to be pre-charged.

The CAM cell was redesigned, as shown in Figure 6.13, so that the CAM always begins an operation with *Bit* and $\overline{Bit}$ pre-charged. This eliminates the delay and unnecessary energy cost of switching the polarity on the bitlines. In addition, the CAM cell is now DVS compatible.

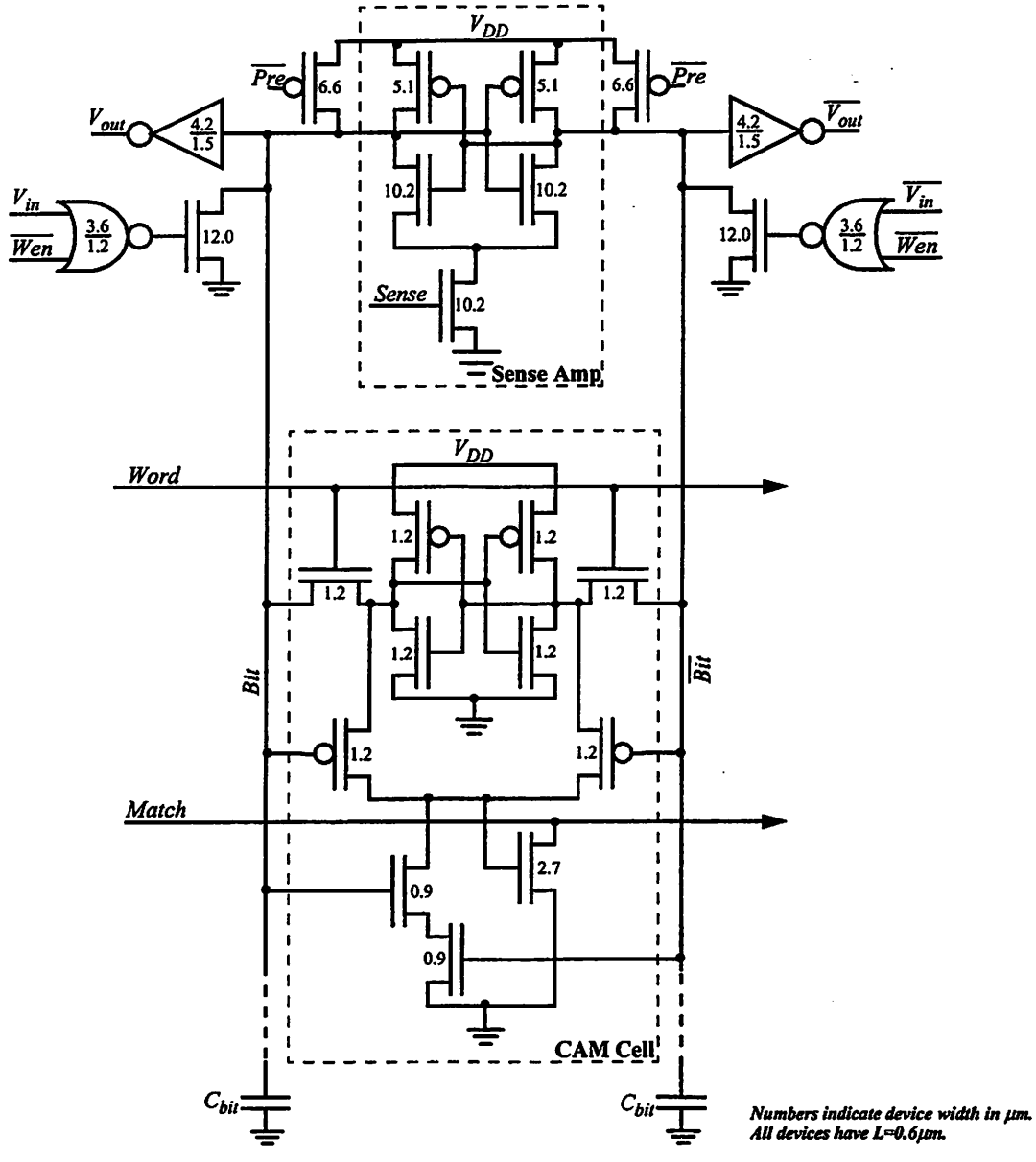


FIGURE 6.13 : CAM Cell and Sense Amp.

The additional five transistors increased the size of the memory cell so that the sense-amp was better pitch-matched to a single memory cell in the CAM, as compared to two memory cells in the SRAM. This change eliminated the need for a column multiplexer so that the bitlines directly drive the sense amp. However, this forces the bitlines to swing full-rail upon a CAM read. Since the CAM is most commonly performing match operations, thereby making CAM reads infrequent, the overall increase in system energy consumption is negligible.

The CAM memory utilizes the same sense-amp and self-timed circuitry as the SRAM, so that its delay has the same characteristics as the SRAM. The delay tracks static CMOS logic very well at constant voltage, but suffers the same deviation when V_{DD} varies during the sensing period.

6.2.3 Register File

The ARM8 architecture requires a 30x32b register file with one write-port and two read-ports. The three ports are independently operated, requiring three different ports to the register cell itself. To reduce the requisite routing overhead, the ports were implemented with single-ended pull-down circuits, as shown in Figure 6.14. The single-

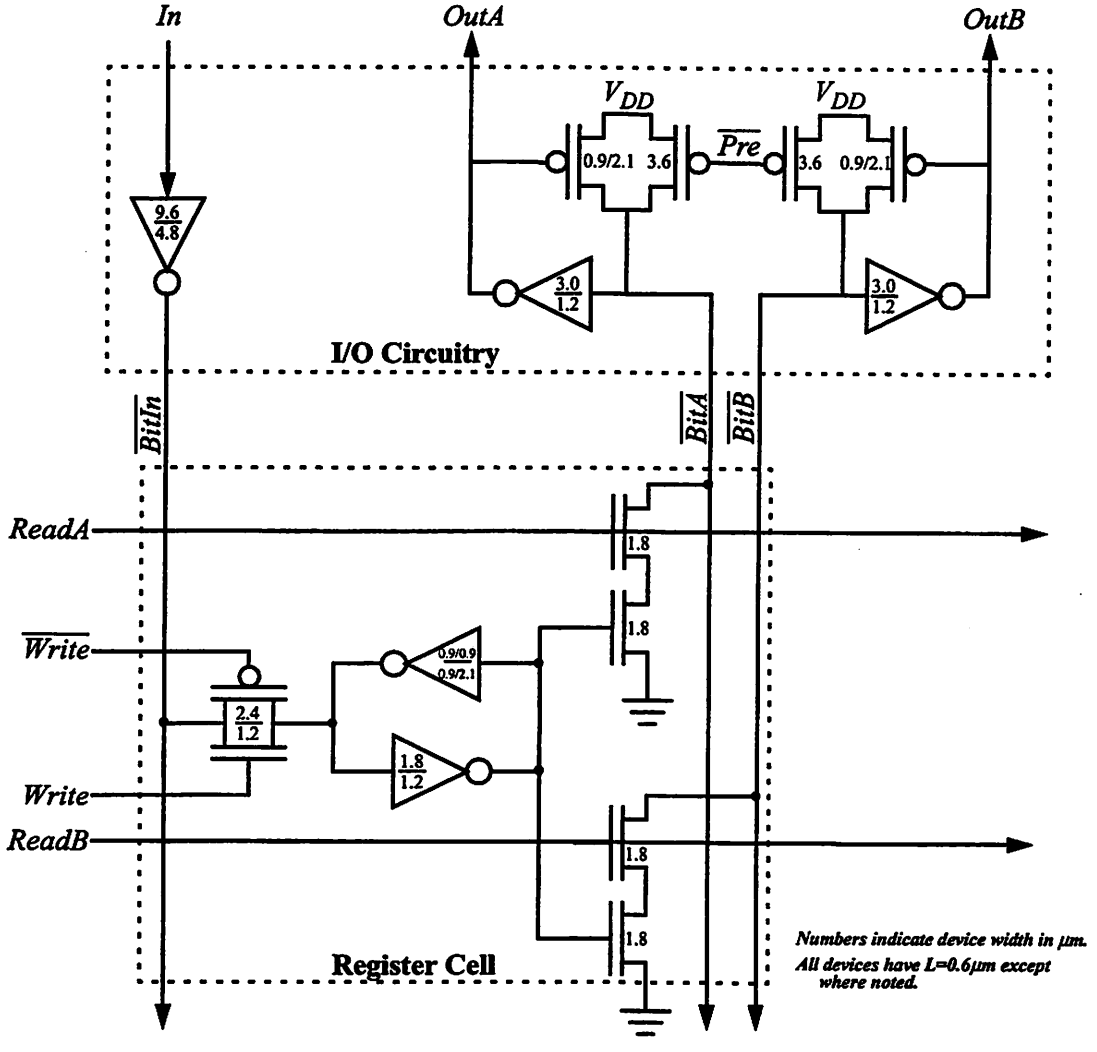


FIGURE 6.14 : Register File Cell and I/O Circuits.

ended ports forced both the input and output bitlines to swing full-rail, but this ensures that the delay of the register file scales over V_{DD} similar to static CMOS logic.

Both *BitA* and *BitB* are pre-charged high, and are selectively pulled down if both the cell's read signal (*ReadA* or *ReadB*) is high and the internal cell state is high. The input data is inverted, because simulation demonstrated that the majority of data bits written to the register file are low. This reduced the energy consumption of the register file datapath by 55%, and the overall register file energy consumption by 33%. The weak feedback transistors are required to maintain state on *BitA* and *BitB* when it is not actively being pulled down in the evaluation state.

6.3 Low-Swing Bus Transceivers

The energy required to drive large busses has become increasingly significant as process technology has improved. While average gate capacitance and local interconnect capacitance have reduced with improved process technology, global bus capacitance has not. Global busses include both intrachip busses, and the interchip, board-level processor bus.

Global intrachip bus capacitance has actually increased with improved process technology, because both wiring capacitance per unit length and average bus length increase with improved process technology. The wiring capacitance grows due to thinner oxides and narrower wiring pitches. Larger die sizes necessitate larger bus lengths for connecting up the various microprocessor blocks. The deployment of copper interconnect has enabled a reduction in the wiring capacitance per unit length, because it could be manufactured thinner than more traditional aluminum interconnect while maintaining the same resistivity. But as copped interconnect is pushed into more advanced process technologies, the wiring capacitance per unit length will continue to once again increase.

Interchip busses, primarily the external process bus, have a capacitance that has remained roughly constant, because it is dominated by printed circuit board capacitance, and packaging parasitic capacitance. However, bus frequencies have greatly increased, driving up the energy consumption due to fast signal edges.

Transceivers for both intrachip and interchip busses are presented, along with measured results from a test chip. These transceivers were not integrated into the prototype system, in order to aid debugging, but integration into a future processor system is discussed. The transceivers were designed to be DVS compatible so that they could be integrated into a DVS processor system.

The key enabler of these low-swing transceivers is the demonstration of a high-efficiency, low-voltage regulator [stra98]. The output voltage can be as low as 200mV with a conversion efficiency in excess of 90% using a standard 3.8V lithium-ion battery for the input voltage.

6.3.1 Intrachip Transceivers

The on-chip transceiver was designed with differential signal lines. This eliminates the need for a reference voltage at the receiver, and makes the bus significantly more immune to coupling from adjacent, interfering signal lines. The penalty is that the bus requires twice as many signal routes, but as the number of metal layers continues to increase with process technology, this penalty is mitigated.

The transceiver architecture is shown in Figure 6.15. The driver uses two pairs of two NMOS devices to drive the bus signals *bit* and $\overline{bit}$ to either V_{LOW} or ground. Since the driver's NAND and NOR enabled gates are powered by the variable voltage V_{DD}, the driver current and delay scales with V_{DD}. The voltage on *bit* and $\overline{bit}$ never exceeds V_{LOW}, which can be little as 200mV, so that the receiver requires a comparator with PMOS inputs to sense the voltage difference. To minimize energy consumption, the comparator utilizes a clocked, dynamic design. The self-timed precharge signal

simplifies the receiver control, which only requires a single clock signal to control it.

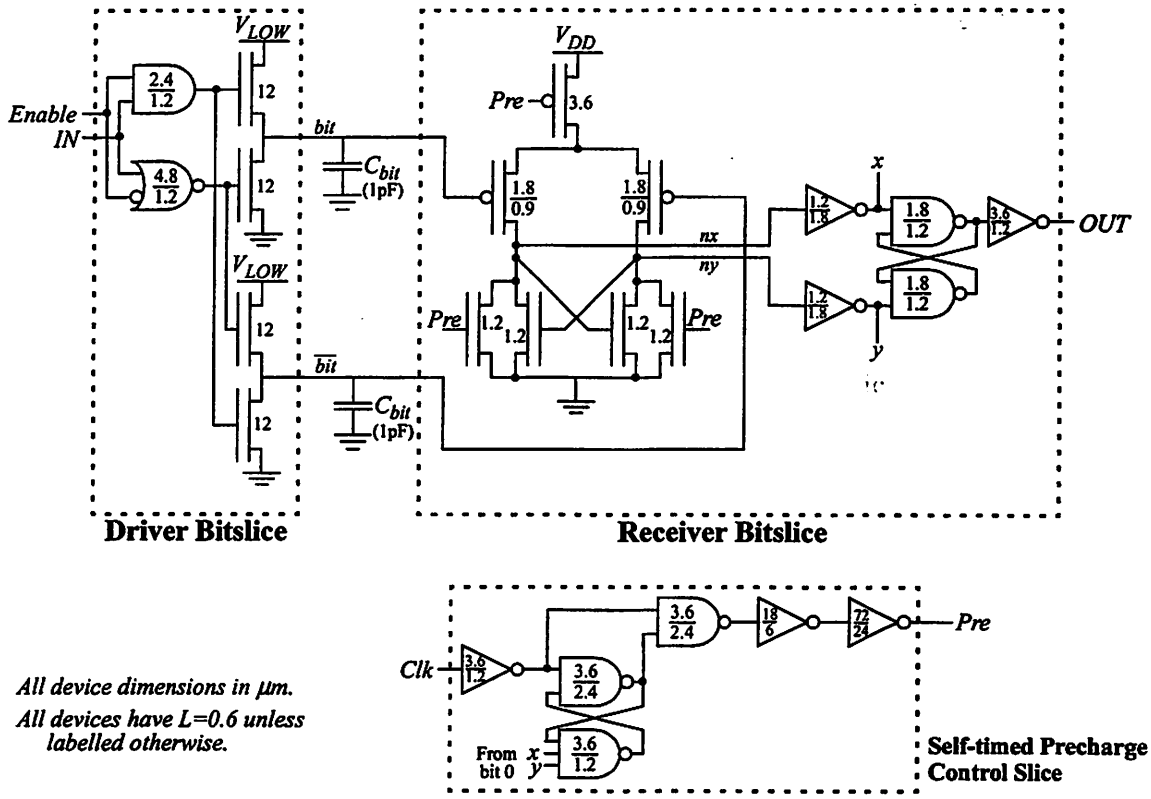


FIGURE 6.15 : Differential Low-Swing Bus Transceiver Circuit

The $\overline{RS}$ latch at the output of the comparator registers when one of the comparator's outputs, *nx* or *ny*, has gone high, and changes the signal *OUT* accordingly. This eliminates any spurious transitions on the output signal. Since the comparator and the $\overline{RS}$ latch is powered by V_{DD}, the delay through the receiver scales with V_{DD} as well, so that the transceiver delay and energy consumption scales with V_{DD} for DVS compatibility. The self-timed precharge circuit puts the comparator back into precharge mode as soon as a transition is detected in order to minimize energy consumption.

Figure 6.16 compares both the delay and energy consumption (in capacitance/ cycle) of the low-swing bus transceiver and a standard static CMOS bus transceiver when driving a 1pF load. The capacitance/cycle was calculated for random inputs, which yields a 50% probability of a signal transition. While the low-swing transceiver delay is twice as long, the capacitance/cycle reduction varies from 1.7x at high voltage

to over 2.5x at low voltage.

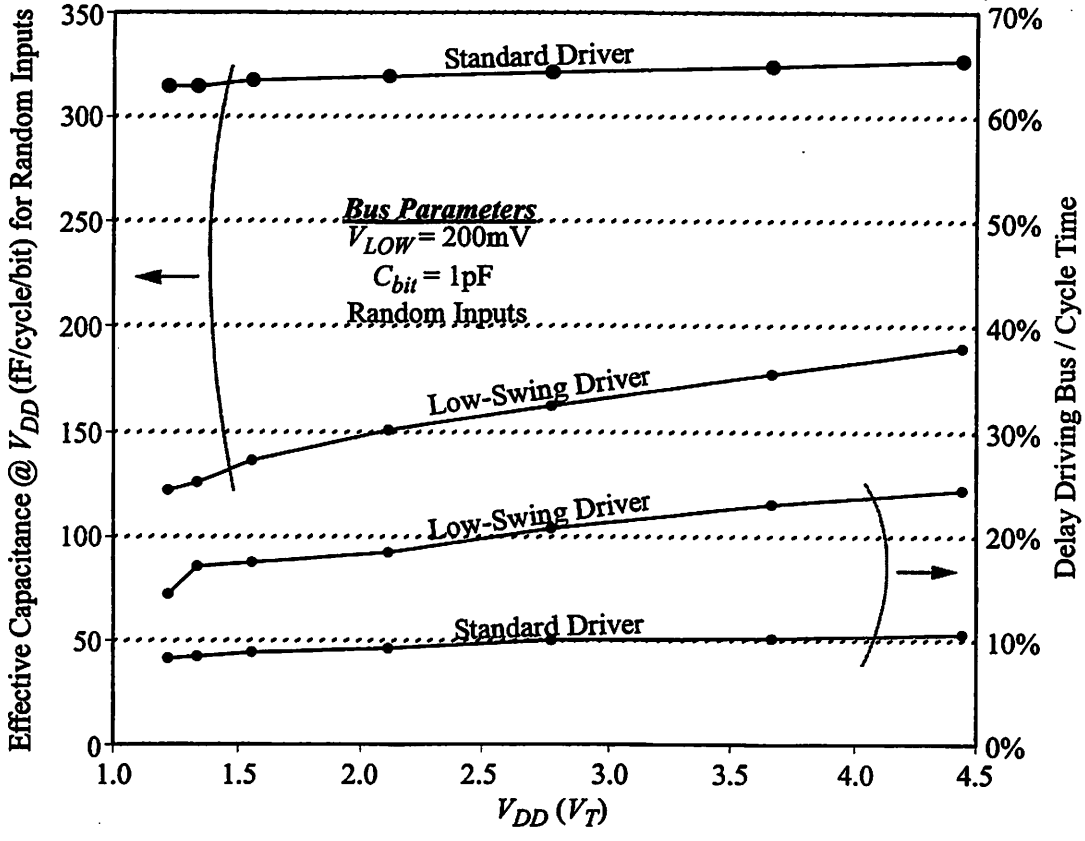


FIGURE 6.16 : Low-Swing vs. Standard Driver for Intrachip Busses (Simulated).

Thus, if architectural design can hide this increased bus latency and place a latch at the receiver, the low-swing intrachip transceiver can provide significant reduction in energy consumption. Additionally, for larger bus capacitances, the capacitance reduction will increase. A 2pF bus capacitance will double the capacitance/ cycle of the standard bus transceiver, but the low-swing transceiver will only increase by 30% at low voltage, and less than 2% at high V_{DD}. This occurs because the bulk of the energy consumed in the low-swing transceiver is by the comparator.

One other limitation to the low-swing transceiver is that while a standard bus driver will only consume energy upon an input signal transition, the low-swing transceiver consumes energy independent of the input signal. Thus, if the data is highly correlated, the low-swing transceiver will actually consume more energy than the

standard bus driver. For the 1pF bus capacitance, if the probability of a transition drops from 50% to 20%, then the standard bus driver becomes more energy-efficient. So in order to evaluate whether a low-swing transceiver can reduce the energy consumption on a bus, both its capacitance and data correlation must be evaluated.

6.3.2 Interchip Transceivers

The off-chip transceiver is a non-differential version of the on-chip transceiver. While the differential signaling provides more robustness, the additional pin-count on the package cannot be tolerated. Thus, the second bitline was removed, and a reference voltage, V_{REF}, set to one-half of V_{LOW} is used as the second input on the differential comparator. The modified transceiver architecture is shown in Figure 6.17. The single-ended receiver does place a constraint on the minimum V_{DD} value:

$$V_{DD} > V_{REF} + V_{Tp} = \frac{V_{LOW}}{2} + V_{Tp} \qquad \text{(EQ 6.8)}$$

so that as V_{LOW} is increased to provide more circuit robustness, the trade-off is decreased operating range on V_{DD}.

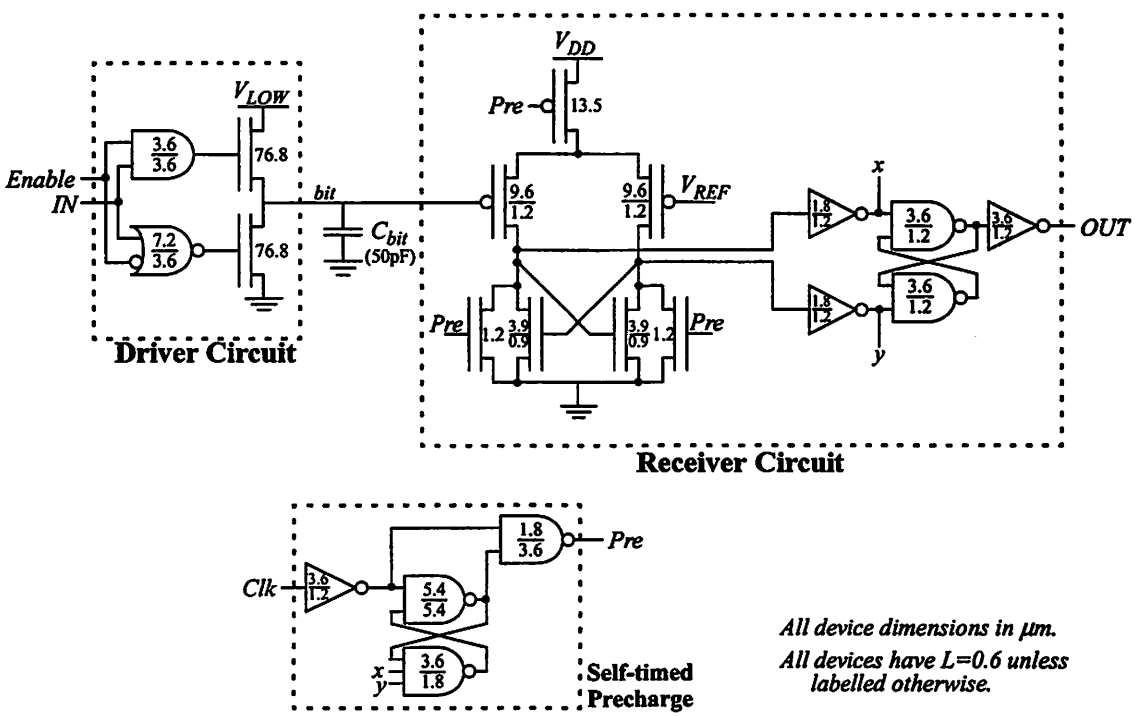


FIGURE 6.17 : Single-ended Low-Swing Bus Transceiver Circuit.

Figure 6.18 compares both the delay and energy consumption (in capacitance/cycle) of the single-ended low-swing bus transceiver and a standard static CMOS bus transceiver when driving a 50pF load. This is a typical capacitance for a bus with ten external chips connected to it, as in the prototype processor system. The capacitance/cycle was calculated for random inputs, which yields a 50% probability of a signal transition. For this case, the low-swing transceiver not only provides slightly less path delay across the bus, but a significant reduction in capacitance/cycle in excess of 15x. Correlation in the bus data will reduce the margin of savings, but the probability of a signal transition would have to be below 3.3% before the low-swing transceiver becomes less energy-efficient. Thus, the low-swing transceiver is extremely well-suited for the external PCB processor bus, which has the key characteristic that the capacitance per bit is anywhere from 25-100pF.

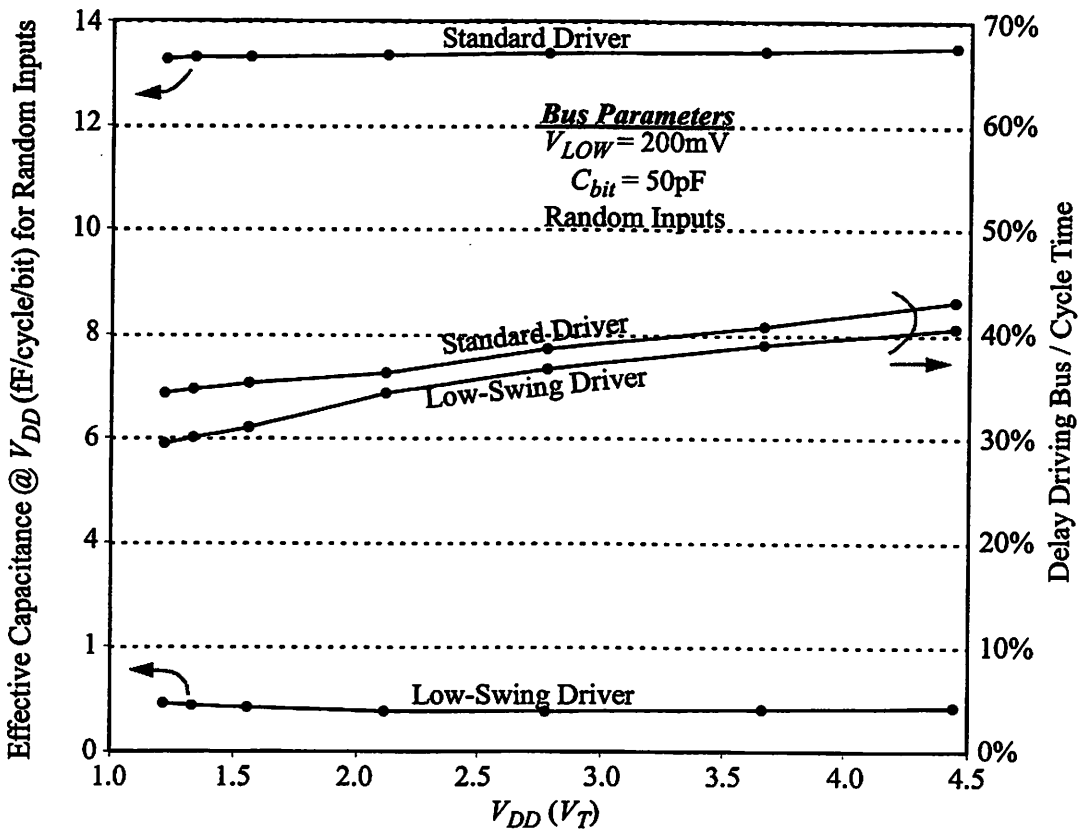


FIGURE 6.18 : Low-Swing vs. Standard Driver for External Busses (Simulated).

A clock signal is still required to drive the external bus, and each cycle it is

required to switch on the order of 50pF of capacitance. Switching the signal at V_{DD} will cause the energy consumed by the clock signal to completely dominate the energy consumed by the low-swing bus. Thus, a continuous transceiver was developed to transmit the clock signal at low voltage to mitigate this component of energy consumption, and this circuit is shown in Figure 6.19.

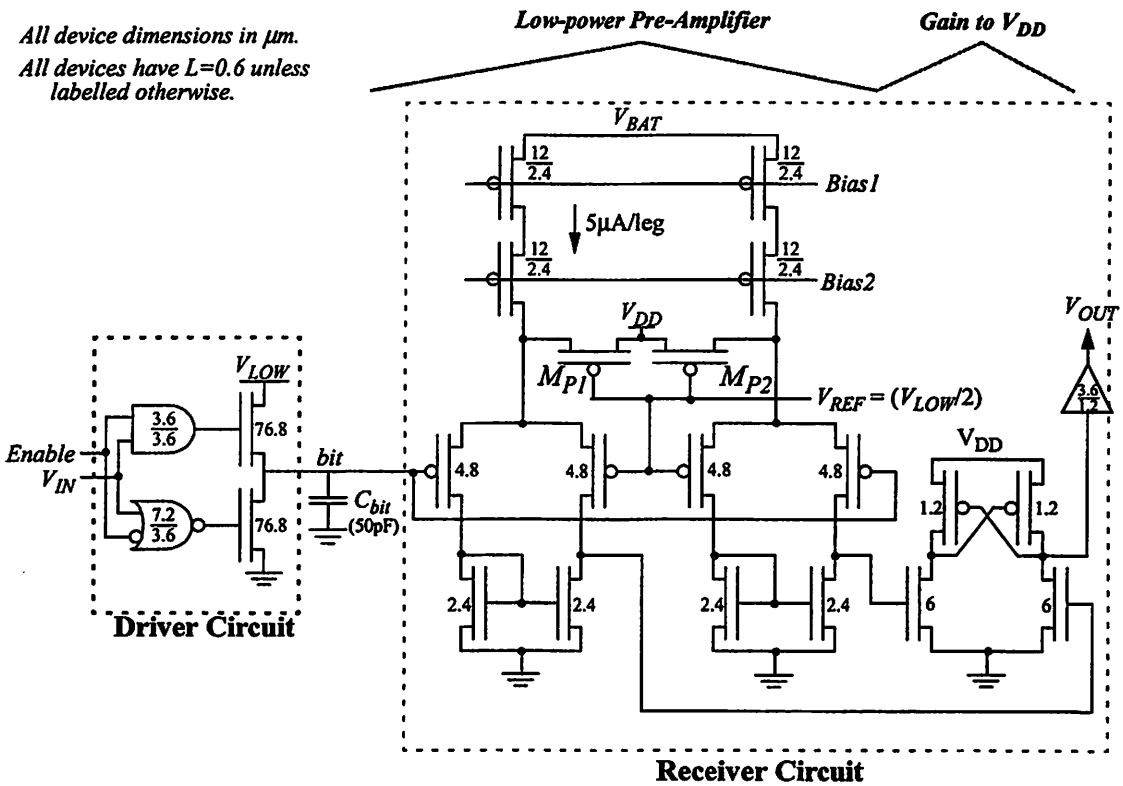


FIGURE 6.19 : Single-ended Low-Swing Clock Transceiver Circuit.

The driver is the same circuit used in the previous transceivers, and converts the input clock signal to a low-swing signal. The receiver consists of two components, the pre-amplifier and the second gain stage. The pre-amplifier's dual source-coupled pair (SCP) circuits convert the signal to differential, and amplifies the signal from 0-V_{LOW} to 0-V_{tp}. The SCP circuits are biased by the battery voltage, V_{BAT}, to ensure a fixed minimum tail-current of 10μA per SCP, which is set by bias voltages, *Bias1* and *Bias2*, from a high-swing cascode current source [gray93]. The devices M_{P1} and M_{P2} provide an additional current source, which is a function of the variable voltage, V_{DD}.

This allows the speed, and the energy consumption of the pre-amplifier to scale with V_{DD}. The second-gain stage amplifies the clock signal to 0-V_{DD}. The cross-coupled loads ensure that this gain stage has no static current, as only one of this gain stage's NMOS devices will be turned on, since $V_{tp} > V_{tn}$. Thus, by using the pre-amplifier to minimize short-circuit current, the receiver provides the necessary signal level-conversion with minimal energy consumption.

The energy consumption in terms of the effective switched capacitance/cycle at V_{DD} is shown in Figure 6.20. A clock signal switching at a voltage V_{DD} would have 50 pF/cycle, while the low-swing clock transceiver has reduced this to less than 2 pF/cycle. The capacitance/cycle is roughly constant, due to the variable-current tail-source in the pre-amplifier. The penalty of using the continuous-time clock transceiver, rather than the previous transceiver with the dynamic latch, is approximately 2x.

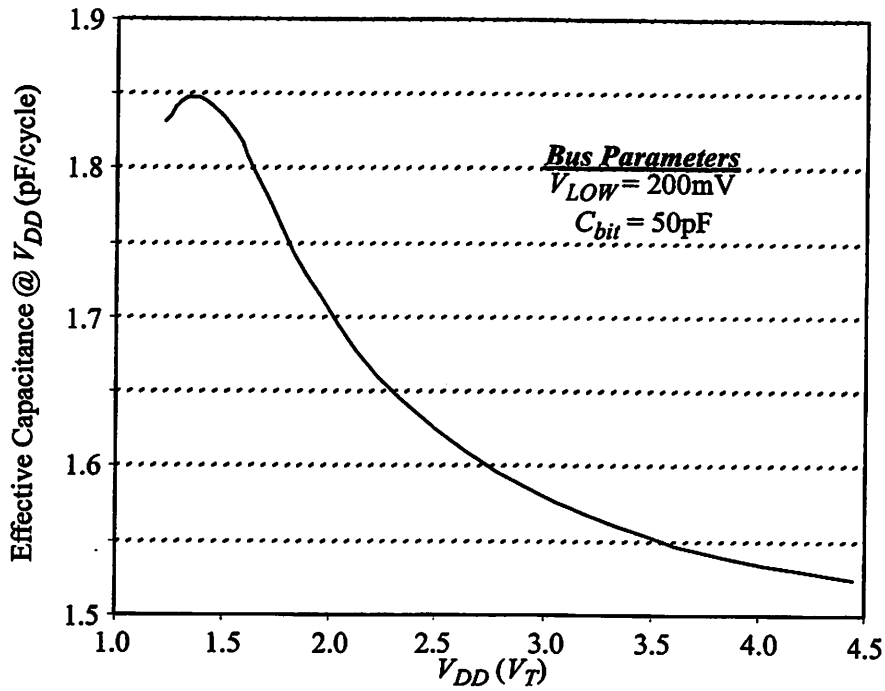


FIGURE 6.20 : Low-Swing Clock Transceiver Energy Consumption (Simulated).

6.3.3 Test Chip

A test chip was fabricated in our 0.6μm process to validate the low-swing transceiver designs. The intra-chip receiver successfully operates with V_{LOW} = 200mV

as V_{DD} ranges from 1.0-4.2V, and the clock frequency ranges from 4-111MHz. For the 1.4pF bus (measured) on-chip bus, the reduction in power dissipation ranges from 1.7x to 3.3x as demonstrated in Figure 6.21, which yields an equivalent reduction in energy consumption. In addition, the intrachip transceiver can operate as V_{DD} varies at a rate of 10 V/μs, demonstrating that this design is compatible with a DVS processor system. At V_{DD} = 1.0V, V_{LOW} can be operated as low as 40mV, though the minimum V_{LOW} for all values of V_{DD} is 150mV.

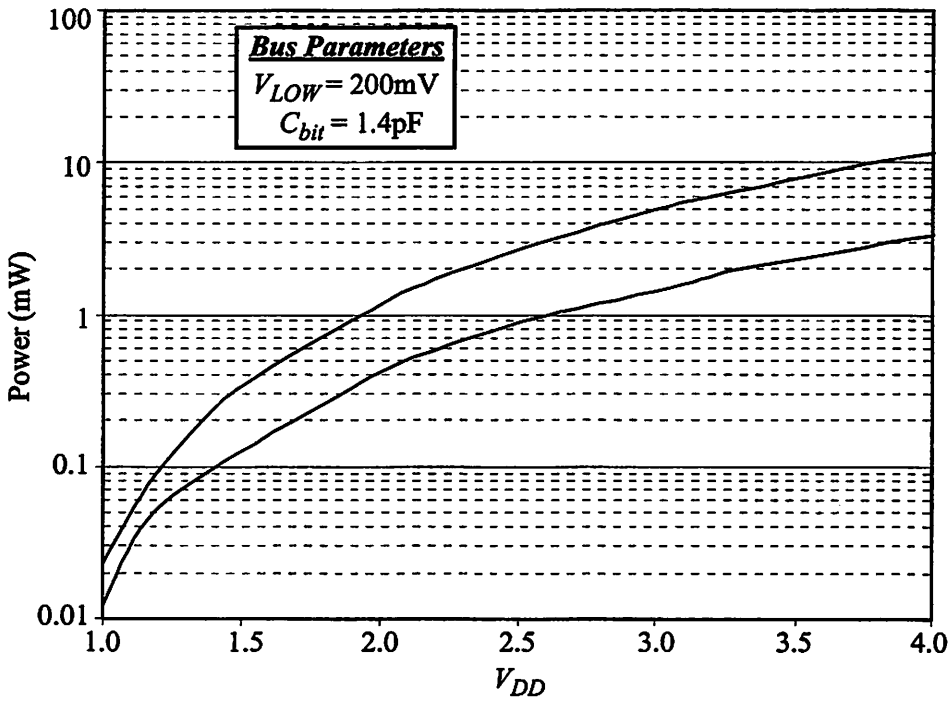


FIGURE 6.21 : Power Dissipation of Low-Swing and Standard Driver for Intrachip Busses.

The inter-chip receiver successfully operates on a 50 pF/bit bus with V_{LOW} of 200mV as V_{DD} ranges from 1.0-3.75V, and the clock frequency ranges from 4-100MHz for the worst-case condition when all bits are switching simultaneously. Higher values of V_{DD} increase the current draw from V_{LOW}, and the resulting noise prevents the receiver from continuing to operate properly. If V_{LOW} is increased to 500mV, the receiver can operate over the range 1.25-3.75V. When the number of bits that switch simultaneously is reduced the receiver can operate at a value of V_{DD} as high as 4.75V due to the decreased current draw on V_{LOW}.

In addition, when all bits are simultaneously switching, V_{LOW} is 500mV, and V_{DD} varies at a rate of 16 V/µs, the receiver can operate over the V_{DD} range 1.25-3.25V. As V_{LOW} decreases to 200mV, the range of V_{DD} drops to 1.0-2.6V. Decreasing dV_{DD}/dt to 1 V/µs allows proper operation over the same range of V_{DD} as when it is held constant. These results are summarized in Figure 6.22.

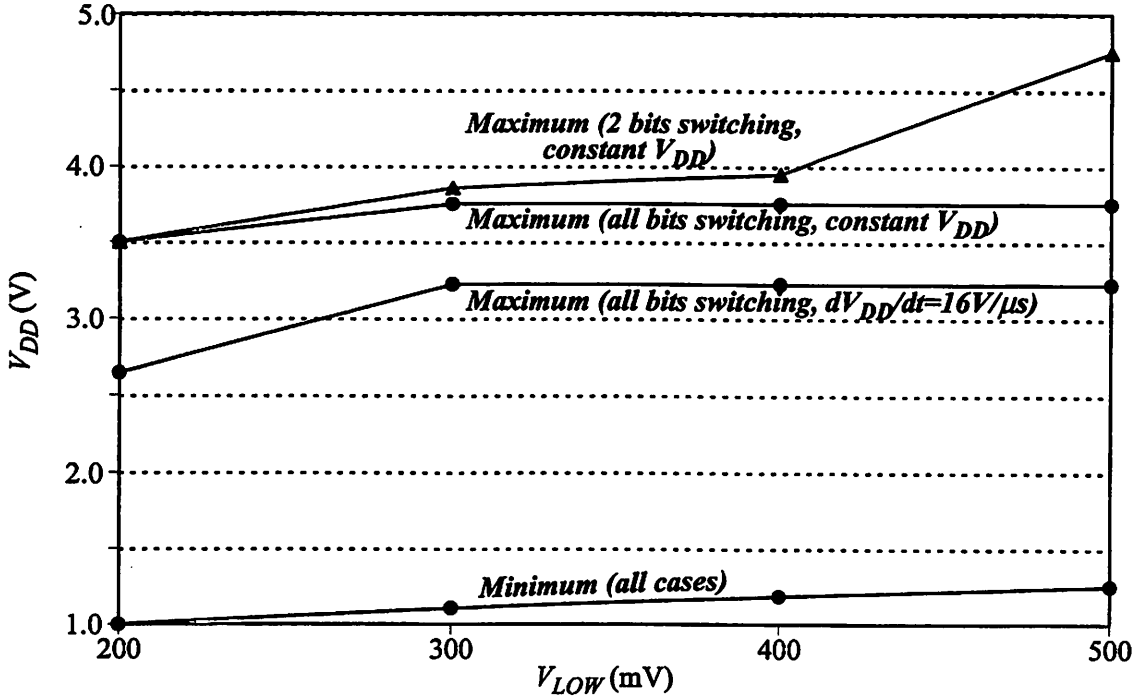


FIGURE 6.22 : Range of Operation for Low-Swing Interchip Bus.

The prototype low-swing inter-chip bus was designed with one power/ground pin per eight bus pins. It is believed that by decreasing this ratio the range of operation for V_{DD} can be increased. In addition, further on-chip bypass capacitance for V_{LOW} will reduce the on-chip noise of this signal, and also improve circuit robustness and operating range.

6.3.4 Future Integration

The prototype chip demonstrates the viability of the low-swing bus transceivers, as well as significant reductions in energy consumption, particularly for the external PCB processor bus. Extra overhead is required for the additional voltage

regulator to generate V_{LOW}, but this can mitigated by sharing circuits between this regulator and the existing system voltage regulator [stra98].

It is believed that additional pins for the low-voltage power and ground will alleviate the headroom problems observed on the test chip, but this requires further investigation. Additionally, for the inter-chip transceiver, it is necessary to provide a stable V_{REF} signal on-chip. For the test-chip, V_{REF} was generated externally with a resistor divider. However, this needlessly dissipates static power dissipation. Further investigation is required to minimize this static power. One possible solution is to generate V_{REF} via a switched capacitor network. Another solution is to use large bypass capacitors either on-chip or internal to the package to maximize the size of the resistors used in the divider.

7

Prototype Microprocessor System

A complete embedded microprocessor system was designed and implemented in 0.6μm CMOS to validate the processor system design methodology described in the previous chapters. By combining Dynamic Voltage Scaling with energy-efficient architecture and circuit design, the system is able to demonstrate more than an order of magnitude improvement in energy efficiency.

In order to measure the energy efficiency of programs typically running on portable devices, a complete software infrastructure was developed. This infrastructure includes a pre-emptive multi-tasking real-time operating system providing standard C library functionality, which allowed standard C programs to be compiled for the system. A programmable I/O board enabled rapid prototyping of I/O devices to verify the system's functionality, and enabled multimedia programs with real-time constraints to run on the system.

Sections 7.1-7 describe the four chips as well as the physical board implementation. Sections 7.8-9 describe the I/O board and the software infrastructure. Section 7.10 presents the measured performance of the prototype system, and Section 7.11 compares this system to prior art and other energy-efficient processors currently available today.

7.1 System Architecture

The prototype system, shown in Figure 7.1, contains four custom chips fabricated in a 0.6μm 3-metal CMOS process technology [hp95]. The chip-set includes a microprocessor, a battery-powered DC-DC voltage converter, a bank of SRAMs, and an interface chip for connecting to commercial peripheral devices, which are modeled by the I/O board. These chips integrate all the necessary logic for inter-chip communication so that they can be seamlessly connected together. For a completely functional processor system, the only external components required are a crystal oscillator, an inductor, and several small bypass capacitors.

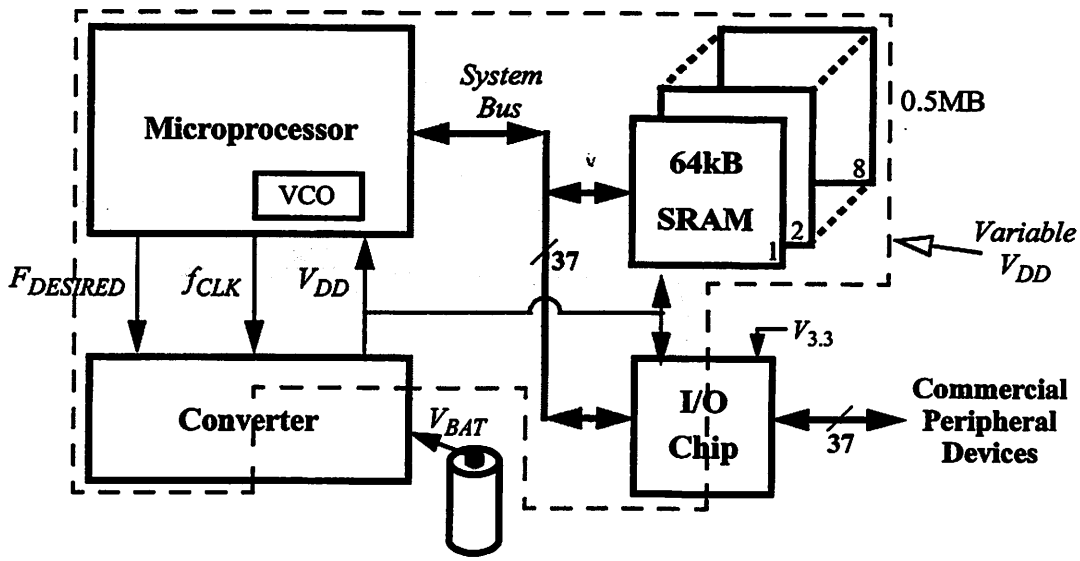


FIGURE 7.1 : Prototype System Architecture.

The DVS voltage regulation loop consists of the battery-powered converter chip, and the VCO which is connected to the loop via the V_{DD} and f_{CLK} signals. The processor commands the desired clock frequency via the digital $F_{DESIRED}$ signal. The 37-bit *System Bus* consists of a 32-bit multiplexed address/data bus, and five bits of control. In addition, the processor generates chip enable signals for the I/O and SRAM chips.

There are three voltage domains in the system. The converter outputs the variable DVS voltage, V_{DD}, which powers the processor, the SRAM chips, the I/O chip, and the front-end circuits on the converter chip. The battery voltage, V_{BAT}, supplies the converter's power FETs and back-end circuits. The 3.3V voltage, $V_{3.3}$, supplies the output pads of the I/O chip so that it can replicate the system bus at a standard voltage level, which allows the bus to connect to commercial ICs.

7.2 Microprocessor IC

The chip's processor core implements the ARM V4 instruction set architecture (ISA) [arm96a]. The implementation was derived from an RTL behavioral model (provided by ARM Ltd.) which fixed both the ISA as well as the processor core interface. However, both the custom physical implementation of the core, as well as the rest of the microprocessor design, were fully optimized for energy efficiency.

Full compatibility of the ISA was critical so that commercially-available compilers, assemblers, and simulators could be used, thereby allowing rapid software development on the hardware platform, and eliminating the need to develop custom software tools. While the microprocessor implementation is ARM-based, the design methodology outlined in the previous chapters is equally applicable to other ISAs, and will yield similar improvement in energy efficiency.

Inter-chip communication is much more costly than intra-chip communication, in terms of both performance and energy efficiency, so that integrating as much system functionality as possible on the microprocessor chip will yield a more energy-efficient implementation. As such, all system logic was integrated onto the microprocessor, with the exception of the main memory, the voltage regulation loop, and the I/O interface. The main memory remained separate because sufficient amounts of memory cannot be integrated onto the same die. The loop and interface remained separate for design simplicity, without having a significant impact on energy efficiency.

The microprocessor die, shown in Figure 7.2, measures 7.5 x 9.0mm and contains 1.3M transistors of which 890k are memory (CAM & SRAM) transistors.

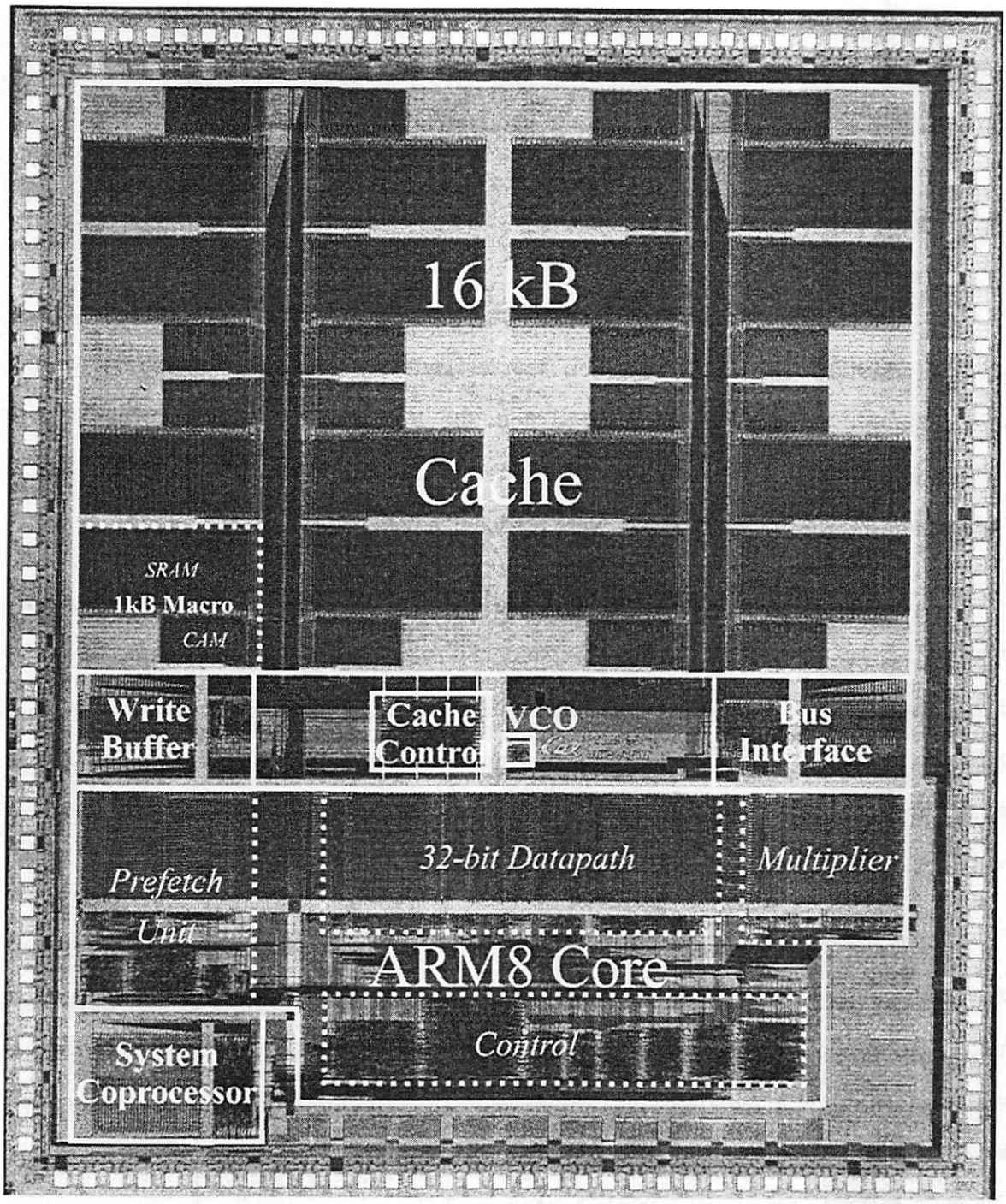


FIGURE 7.2 : Microprocessor Die Photo.

7.2.1 Architecture

A high-level block diagram of the microprocessor's architecture is shown in Figure 7.3. The processor core is a custom implementation of an ARM8, which is a 32-bit five-stage scalar integer pipeline with an eight-word prefetch unit that performs static branch prediction [arm96b]. The system coprocessor contains system control state which can be manipulated by the ARM8 core via its coprocessor interface. The cache sub-system consists of a 32-way set-associative 16kB memory, a twelve-element write

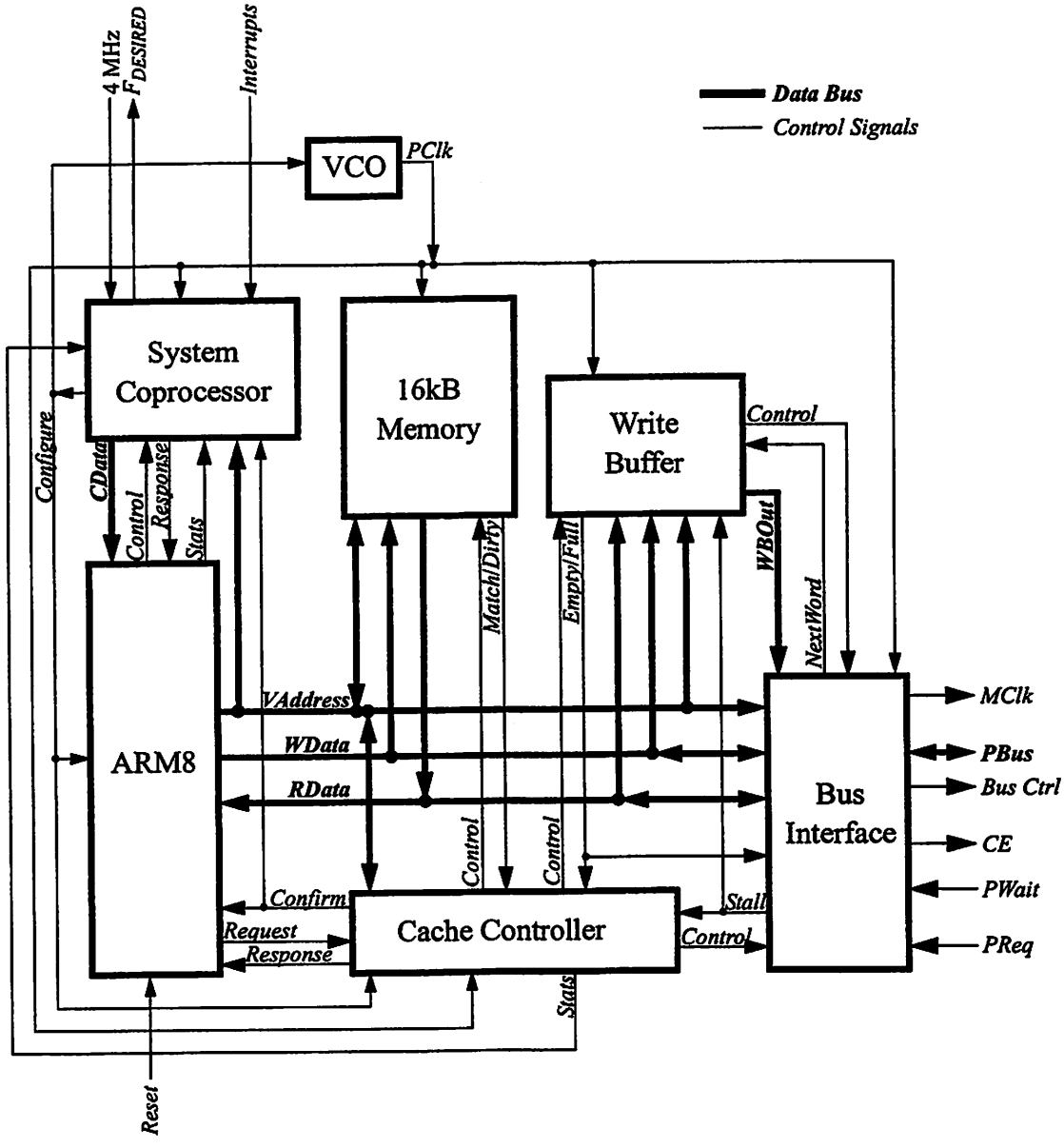


FIGURE 7.3 : Microprocessor Architecture.

buffer, and a bus interface, all of which are managed by the cache controller. The cache is physically indexed to eliminate the need for a TLB, and the upper six bits of the address are utilized for memory-space control which gives the microprocessor an effective 26-bit address space. The bus interface drives the external system bus, and contains a memory controller which allows external memory chips to be seamlessly connected to the bus. The VCO provides the variable-frequency clock signal, *PClk*, to all the internal microprocessor components, and *PClk* is buffered and transmitted to the converter as the f_{CLK} signal. The external bus is clocked by *MClk*, which is divided down from *PClk*.

7. 2. 1. 1 Data Flow

The data flow of the microprocessor was designed around the fixed ARM8 interface. The memory interface consists of three unidirectional busses; the address bus (*VAddress*), the write data bus (*WData*), and the read data bus (*RData*).

The system coprocessor sends data to the ARM8 on a dedicated unidirectional bus (*CData*). The coprocessor receives data via *VAddress*, and during this transfer cycle, the ARM8 must suppress any pending memory access and place the data word on *VAddress*. Since coprocessor writes are infrequent, this has negligible impact on processor performance due to the forced cancellation of memory-access cycles.

VAddress is an input to the write buffer and the bus interface, and is an input/ output to the cache memory so that the tag array can be read and written. The cache controller also reads and writes the lower bits of *VAddress* so that it can detect cache-line boundaries and generate cache memory block enables on reads, and increment *VAddress* across a line for cache-line loads and write-backs.

WData is an input to the cache memory and the write buffer. The bus interface has a bidirectional *WData* port so that it can input non-cacheable ARM8 writes to be sent to external memory, and output data to the cache memory during cache-line loads.

RData is an output of the cache memory and an input to the write buffer. The bus interface *RData* port is also bidirectional so that it can input data from the cache memory during cache-line write-backs to external memory, and send data to the core for non-cacheable ARM8 reads.

Since the system data bus is a single, bi-directional bus (*PBus*) which carries time-multiplexed address and data words on it, as described further in Section 7.4, the bus interface multiplexes the three internal busses onto *PBus*. Due to this multiplexing, the write buffer stores both address and data words into a single twelve-element queue, whose contents are sent to the bus interface via a dedicated unidirectional bus (*WBOut*).

7. 2. 1. 2 Clock Control Domains

To eliminate unnecessary clocking and circuit activity, there are three top-level clock control domains as shown in Figure 7.4. The core domain contains the ARM8 core and system coprocessor, which fetch and execute instructions when this domain is active. The ARM8 expects read and write memory accesses to complete in a single cycle, and communicates these accesses to the cache controller via the ARM8's memory request and response handshake protocol [arm96b]. When the cache sub-system cannot

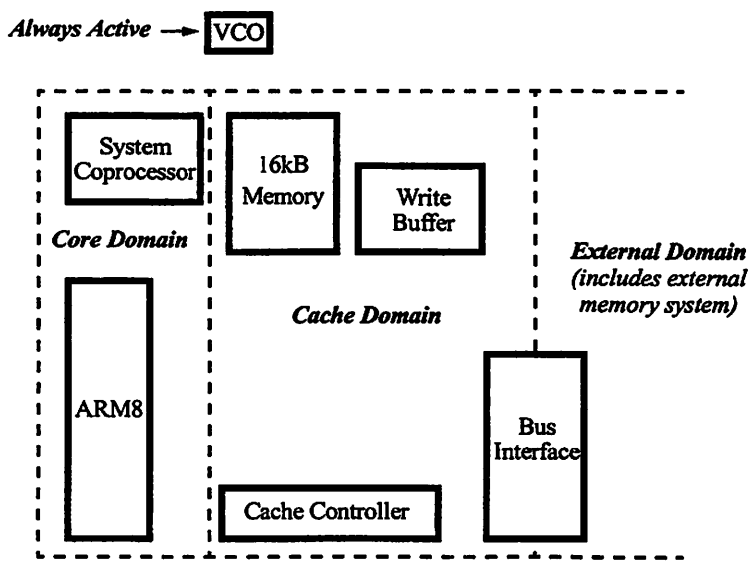


FIGURE 7.4 : Clock Control Domains.

complete the request in one cycle (e.g. cache miss, full write buffer, blocked bus interface, etc.), the cache controller halts the core domain via the *Confirm* signal, which gates *PClk* within all of this domain's clock drivers.

The cache domain contains the entire cache subsystem with the exception of the system-bus side of the bus interface. The cache controller directs the operation of the cache memory, write buffer, and bus interface via various control signals. In response to a memory request from the core, the cache memory's tag array returns whether the request is a match, and if it is not, the cache controller initiates a cache-line load. If the access is not a match and the cache line is dirty (i.e. it has been written to in the cache, but not updated in main memory), a cache-line write-back must be executed first before initiating the cache-line load. The cache controller routes all bufferable writes to the write buffer. When the buffer is full during a pending write, then the cache controller halts the processor and waits for a slot to open. Similar to the ARM8 memory interface, the cache controller expects a read or write access to the bus interface to complete in a single cycle. Since the system bus can operate at no more than one-half the internal clock speed, the bus interface will halt the cache controller, via the *Stall* signal, until the access is complete. This in turn halts the core domain's clock, as well.

The external domain is clocked by *MClk*, and encompasses the system-bus side of the bus interface, as well as the external memory system. An asynchronous interface connects the core-side of the bus interface, which is manipulated by the cache controller and clocked by *PClk*, to the bus-side of the bus interface, which connects directly to the external system bus and generates the external memory chip enables (*CE*). For external memory reads and I/O accesses that require multiple *MClk* cycles, external chips can stall the bus-side of the bus interface via *PWait*, which also halts the first two clock domains as well since they are both waiting for the pending system bus access to complete.

7. 2. 1. 3 Write Buffer Control Flow

When there is no pending external-memory access request from the cache controller in a given cycle, the bus interface polls the write buffer to see if is not empty. When it is non-empty, the bus interface will autonomously initiate a system bus access to write out the data to the external memory system. During this transfer, if the cache controller has an access request to the bus interface (e.g. non-cacheable memory request from the ARM8, cacheable request that takes a cache miss, etc.), the cache controller must wait for the transfer to complete and stall the core clock domain. To ensure memory consistency without additional hardware, any ARM8 read request must stall until the write buffer is empty [henn95]. With the large 16kB cache, this condition is infrequent and stalling the core has negligible impact on processor performance.

7. 2. 1. 4 DMA Control Flow

The I/O chip can directly access the main memory via a direct memory access (DMA). A DMA request is sent to the microprocessor via the *PReq* signal. When there is no outstanding system bus transfer, the bus interface grants the DMA request and releases control of the system bus to the I/O chip. Until the DMA request completes, any access request to the bus interface from the cache controller is stalled.

7. 2. 1. 5 Processor Configuration & Monitoring

The system coprocessor, described further in Section 7.2.6, is responsible for configuring the microprocessor and collecting dynamic statistics. The coprocessor sets the processor speed by transmitting $F_{DESIRED}$ to the converter chip, and controls the voltage-to-frequency conversion by configuring the VCO. In addition, it can configure the ARM8 core, as well as the operation of the cache subsystem via the cache controller. The coprocessor collects run-time statistics from both the ARM8 and cache controller, which can be accessed in software via a coprocessor read instruction.

7.2.2 Processor Core

The processor core is a fully-compatible, custom-implementation of the commercial ARM8 core. Starting from a block level RTL behavioral description, the design methodology described in previous chapters was utilized to provide an energy-efficient and DVS-compatible implementation. This section provides an overview of the ARM8 core and highlights some of the specific design optimizations. A more detailed description of the core's functionality, I/O interface, and signal timing can be found in the ARM8 data-sheet from ARM Ltd. [arm96b]

7. 2. 2. 1 ARM8 Instruction Set Architecture

The ARM8 is similar to a traditional RISC ISA as it is a load-store architecture. Data processing instructions can only operate on registers; external memory contents can be loaded to and stored from the register bank, but not operated on directly. In addition, the instructions are a fixed size of 32 bits. These characteristics allow the ISA to map onto the common five-stage pipeline found in simple RISC processor cores.

There are some non-traditional features of the ARM8 ISA which prove useful in embedded applications by reducing the machine code size. However, these add complexity to the pipeline control and data flow. The primary features are:

- All instructions are conditionally executed. Each instruction has a four-bit condition field, which must be evaluated before writing the results of the instruction to the register bank or main memory, or passing the results to subsequent instructions via data forwarding.
- The second operand of data processing instructions can be shifted before the data processing operation. A five-bit field specifies the shift amount, and the shift type can be logical left, logical right, arithmetic right, and rotate right. Because of this feature, there is no explicit shift instruction.

- Block data transfers to and from memory. Unlike regular load/store instructions which operate on a single register, block transfers operate on a list of registers as specified by a 16-bit field. This is a multi-cycle operation which halts subsequent instructions in the pipeline until the operation has completed.
- Multiply and multiply-accumulate instructions. They require special hardware to implement, and impact data flow because of the 64-bit result generated by a 32 x 32 multiply and the need for the result to pass through the ALU to perform the accumulate.
- Complex addressing modes. The ISA supports both immediate address offsets from the base address register, and register-shifted offsets. In addition, the ISA allows pre/post indexing and auto increment/decrement addressing modes [henn95].

7. 2. 2. 2 ARM8 Pipeline

The basic ARM8 pipeline has five stages, as shown in Figure 7.5. However, the *Write* stage is only used by load instructions when writing data to the register file. All other instructions (e.g. data processing, store, branch, coprocessor) complete by the end of the fourth stage at which time they have completed any writes to the register file. The coprocessor operates lock-step with the ARM8 pipeline, but with a half-cycle delay, which is described in more detail in Section 7.2.6.

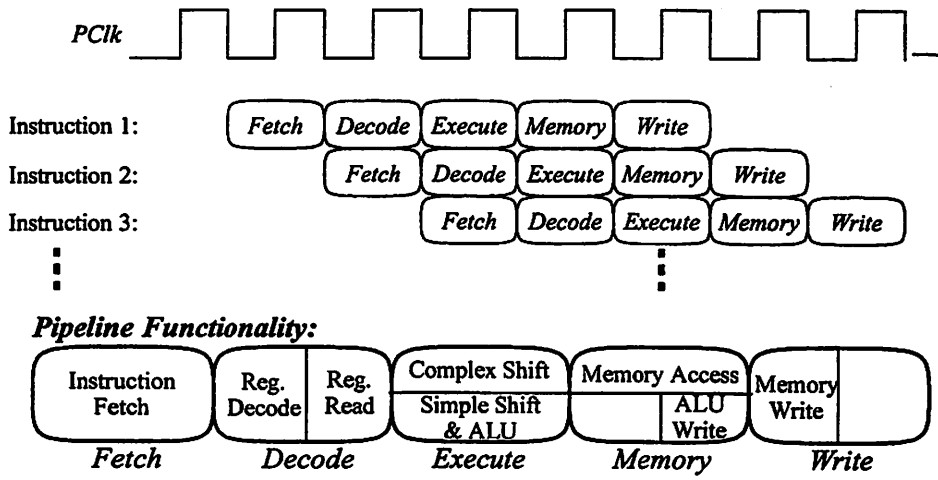


FIGURE 7.5 : ARM8 Pipeline.

Complex instructions will cause the core to iterate on a pipeline stage more than once, forcing all previous pipeline stages to halt. Simple shifts (left-shift by zero, one, two, or three bits) are optimized to complete in the same cycle as any operation requiring the ALU, but complex shifts, which require the use of the barrel shifter, force the core to loop on the *Execute* stage twice - one full cycle for the shift, and another full cycle for the ALU operation. The multiply instructions will cause the core to loop on the *Execute* stage a variable number of cycles, depending on the operands' data. Block data transfers will force the core to loop on the *Memory* stage until all the requisite registers have either been loaded or stored.

7. 2. 2. 3 ARM8 Data Flow Architecture

Figure 7.6 present a block diagram of the ARM8 and highlights the main sub-blocks of the core's datapath. The prefetch unit fetches instructions from the memory via *RData* and places them in an eight-deep FIFO. The memory address is generated by the program counter (PC) incrementer block and placed on *VAddress*. Whenever there is no load or store pending on the memory bus, and the FIFO is not full, the prefetch unit will fetch more instructions from memory. The datapath fetches instructions from this FIFO in the first (*Fetch*) pipeline stage, and decodes the instruction in the first half of the *Decode* stage.

In the second half of the *Decode* stage, the register operands are read from the register file and placed on *ABus* and/or *BBus*. At the beginning of the *Execute* stage, these busses input data to the multiplier, the write-data pipeline, or the ALU. The multiplier will stall the datapath for two to six additional cycles, depending upon how many of the most-significant source-operand bytes are zero, and place the output product back onto *ABus* and/or *BBus*. The product passes through the ALU to be written back into the register file. The write-data pipeline holds the register's data to be saved for one cycle until the *Memory* stage, at which time the data is then placed on *WData*. Simple ALU operations will complete in one cycle during the *Execute* stage, place the

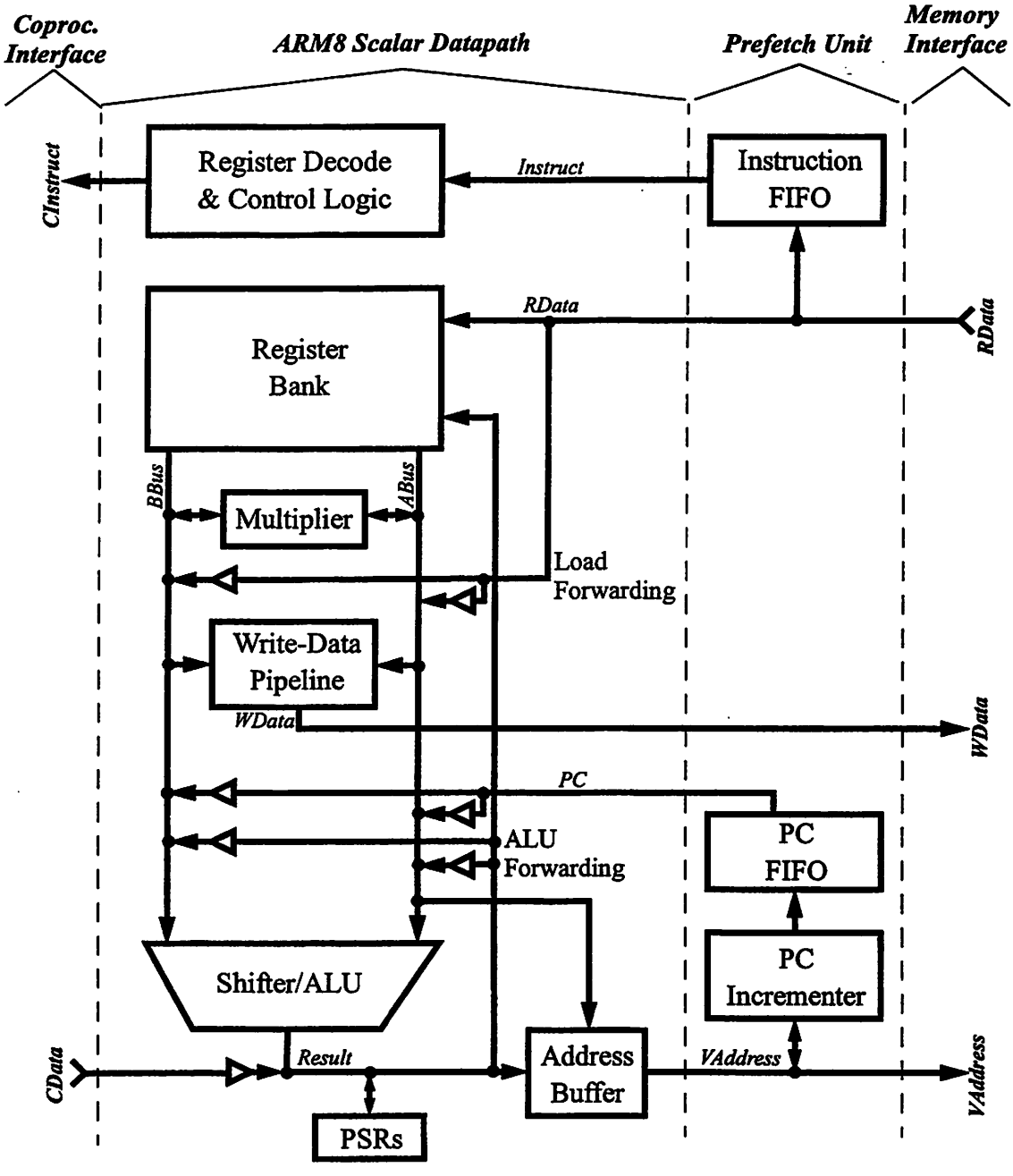


FIGURE 7.6 : ARM8 Data Flow Block Diagram [arm96b].

result on the *Result* bus, and write the value back to the register file during the second half of the *Memory* stage.

For loads and stores, the ALU is used to calculate the effective memory address during the *Execute* stage, and the address is sent to the address buffer via the *Result* bus. The address is then placed onto *VAddress* during the *Memory* stage. Stores

will complete in this stage by placing the stored data onto *WData*. Loads will be initiated during this stage, but the memory data will not be valid until the end of this stage, and written to the register file in the first half of the *Write* stage.

When the PC is used as an operand register, the PC FIFO is used to hold the value until the end of the *Decode* stage, at which time it is placed onto either *ABus* or *BBus*. Writes to the PC are done via *VAddress* and flush all previous instructions in the pipeline, inducing a two to four cycle penalty, depending upon the instruction. At the same time the PC is placed on *VAddress,* to be latched into the PC incrementer block, the instruction at that location is fetched and the prefetch unit begins loading subsequent instructions.

To remove read-after-write (RAW) hazards in the pipeline, there are two sets of data-forwarding paths. The ALU-forwarding path can bypass the *Result* bus to *ABus* and *BBus* when a data processing or effective-address calculation result is used as an operand in one of the next two subsequent instructions. It is either immediately forwarded at the end of the *Execute* stage, or at the end of the *Memory* stage, in which it is simultaneously being written back to the register file. Load instructions do not return their data value to the datapath until the end of the *Memory* stage, so if the very next instruction uses the loaded register as an operand, the datapath must stall for one cycle. To make the data available for the *Execute* stage that coincides with the load's *Write* stage, the load-forwarding path puts the returned data value onto either *ABus* or *BBus* while simultaneously writing the data to the register file.

The 32-bit Process Status Register (PSR), shown in Figure 7.7, contains the

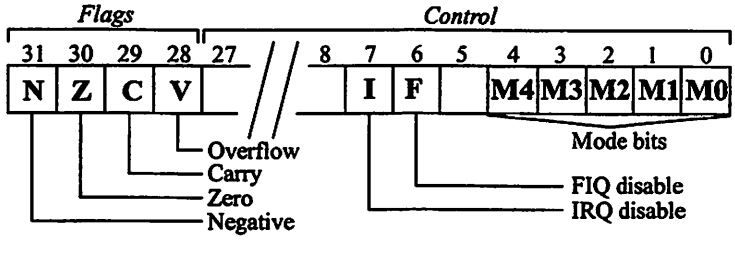


FIGURE 7.7 : Process Status Register.

condition code flags, the interrupts disable flags, and the mode bits which indicate which mode the ARM8 is operating in (e.g. user, supervisor, interrupt). Writing to the PSR depends upon the current operating mode, and is done via the ALU and the *Result* bus. Reading from the PSR is allowed in any mode, and is done by placing the register value onto the *Result* bus and writing it to the register file. Instructions that set the condition code flags do so at the end of the *Execute* stage. Subsequent conditional instructions proceed as normal through the pipeline, but can be flushed in either the *Decode* or *Execute* stage once the value of the flags are known

The ARM8 interfaces to the system coprocessor via three busses. The pre-decoded coprocessor instruction is placed onto *CInstruct* in the first half of the *Decode* stage so that it is available on the rising edge of *PClk*. On this edge, the coprocessor enters its half-cycle delayed decode stage. Reads from the coprocessor to the ARM8 are performed via *CData*, which is driven during the coprocessor's execute stage so that data can be written to the ARM8's register file in the second-half of the *Memory* stage. Writes to the coprocessor are done by placing the data value on *VAddress* during the *Memory* stage, which makes the data available to be written into the coprocessor's register file during its memory/write stage. The coprocessor pipeline is described in more detail in Section 7.2.6.

7. 2. 2. 4 ARM8 Memory Interface

Although a split instruction/data cache structure is generally more energy efficient, the ARM8 memory interface is designed to connect to a unified cache due to legacy system architecture constraints. Since, on average, there is more than one memory read per instruction, due to instruction fetches and loads, the interface is designed to return up to two words per cycle to eliminate this bottleneck. While this would seem to shift the bottleneck to the cache memory's critical path, Section 7.2.3 will demonstrate that by constraining when the cache will return two words, the memory's critical path can be significantly reduced with little or no degradation of

processor performance.

The timing of the memory interface is shown in Figure 7.8. Both *VAddress* and the *ARequest* control signal are driven by the ARM8 when *PClk* is high so that they are available to the cache system on the falling edge of *PClk*. If the memory request is a store, then the data word to be written is placed on *WData* when *PClk* is low. The ARM8 always expects to get acknowledgment of the memory access request from the cache controller by the rising edge of *PClk* via the *AResponse* control signal. For instruction fetches and loads, the ARM8 expects the word to be available by the falling edge at the end of the memory access cycle. If two words are requested, it expects the second word to be available on the rising edge of *PClk* following the access cycle.

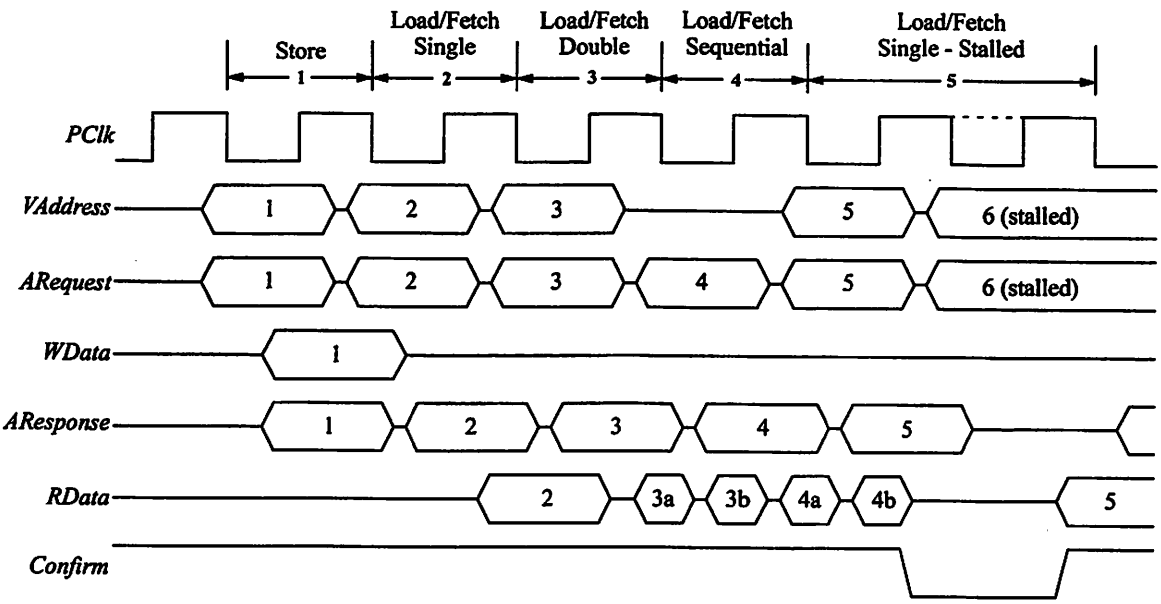


FIGURE 7.8 : ARM8 Memory Interface Timing.

If the cache system cannot complete the access request in one cycle, it must still return an acknowledgment on *AResponse* when *PClk* is low, then deassert the *Confirm* signal when *PClk* is high. Forcing *Confirm* low stops the clock signal in the core clock control domain. The cache system reasserts *Confirm* when *PClk* is high during the cycle it can complete the request, as illustrated for the fifth access sequence in Figure 7.8. Due to the nature of the memory pipeline, the ARM8 will already have placed the sixth access request onto the interface, but it will stall until the fifth access

request has completed.

The ARM8 encodes in the *ARequest* control signal whether the access request is sequential to the last request of similar type (i.e. instruction fetch, data load, data store), whether an instruction fetch is speculative or not, and whether the data load/ store will have more sequential accesses to follow as part of a block transfer instruction. Access requests that are sequential do not need to drive *VAddress* because the cache system can infer the new address by incrementing the previous address. These hints from the ARM8 are utilized to improve the energy efficiency and performance of the cache, which is described in further detail in Section 7.2.3.

7. 2. 2. 5 Optimizations for Energy Efficiency

The ARM8 RTL behavioral model specified the microarchitecture of the core, and was segmented into 29 sub-blocks. In order to use the model's companion test vector suite, which provided vectors for the complete core as well as the individual sub-blocks, the microarchitecture could not be altered. Generating a new suite requires a tremendous effort, and outweighed the potential improvement in energy efficiency that might be achieved by modifying the core's microarchitecture. Thus, only the physical implementation of the processor core was optimized.

Before starting the design, an effective switched capacitance budget for the core was set. The budget is in capacitance, and not energy consumption, because with DVS the energy will vary with V_{DD}, but the effective switched capacitance will remain roughly constant. A budget was necessary to speed up design time so that only those blocks that significantly contribute to the total core capacitance were energy optimized. The design optimizations utilized the circuit design methodology outlined in Chapter 6.

Previous analysis [burd94b] and discussion with ARM (regarding an ARM7 core) indicated that for simple, scalar processor cores, three blocks -- the register file, shifter, and ALU -- contribute more than 50% of the total processor capacitance/cycle.

However, the ARM8 has a prefetch unit, whose additional complexity was estimated to reduce the contribution of these blocks to approximately 33%. Energy-efficient test implementations of the three blocks in the target 0.6µm process technology were 50 pF/cycle, which was then multiplied by three to get the budgeted effective switched capacitance of 150 pF/cycle. This budget was believed to be an aggressive, yet achievable goal.

Previous research has demonstrated that for a large enough sample of machine code, there is little variance in the effective switched capacitance per cycle for scalar microprocessors [peri00]. Since the test vectors for each of the sub-blocks originated from machine code run on the entire core to thoroughly test the sub-block, the capacitance/cycle measured for each sub-block executing its own test vectors, when summed for all sub-blocks, should approximately equal the capacitance/cycle measured for the entire core running typical machine code. This critical observation allowed each of the sub-blocks to be optimized in isolation, greatly reducing simulation time, and yielding an overall energy-optimized processor core because individual sub-blocks can be simulated much faster than the entire processor core.

This is validated in Figure 7.9 which compares the measured capacitance/cycle when simulating the entire core (black) versus when simulating an individual block (white). All of the individual sub-blocks compare to within 20%, with the exception of the last sub-block, the multiplier, because it is exercised much more heavily in the test code than in typical machine code. The 20% maximum variation can be reduced to approximately 10% if the global bus capacitance is modeled in the sub-block simulation, which is not the case for the measured data in Figure 7.9.

To further speed up the design time, the schematics were first energy optimized before the time-intensive task of committing them to custom layout. Figure 7.10 compares the measured capacitance/cycle for each sub-block when simulating the schematic and extracted layout netlists. The relative capacitances compare very well,

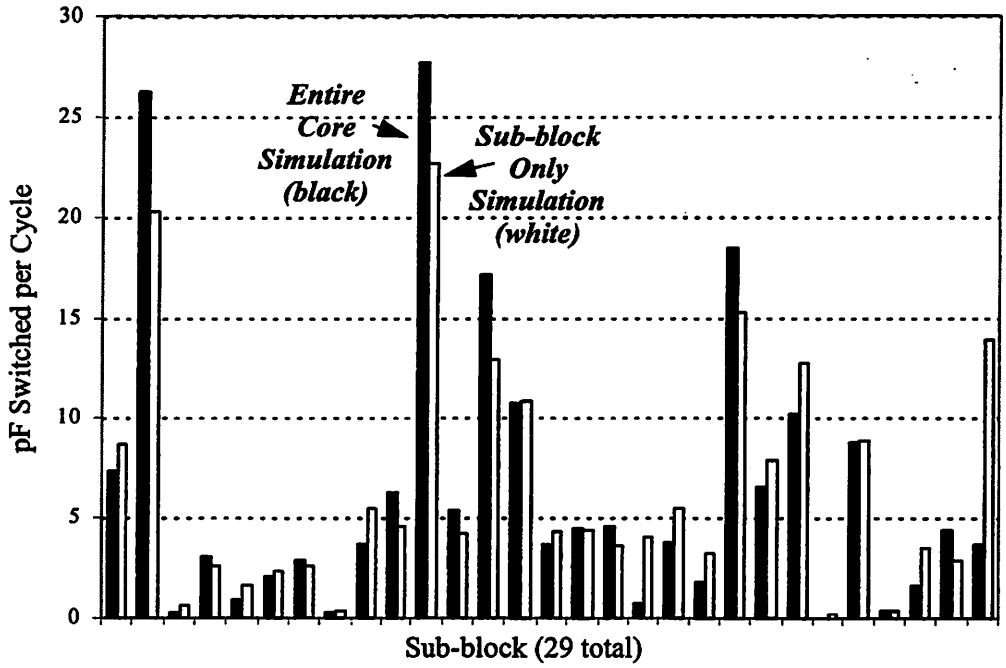


FIGURE 7.9 : ARM8 Capacitance Comparison of Sub-block vs. Entire Core Simulation.

with the extracted netlist yielding slightly higher capacitance due to the inclusion of interconnect capacitance. Only two blocks radically deviate, which are the register file, due to the overestimation of drain capacitance on the bitlines, and the ALU/shifter, due to a large number of busses adding significant interconnect capacitance.

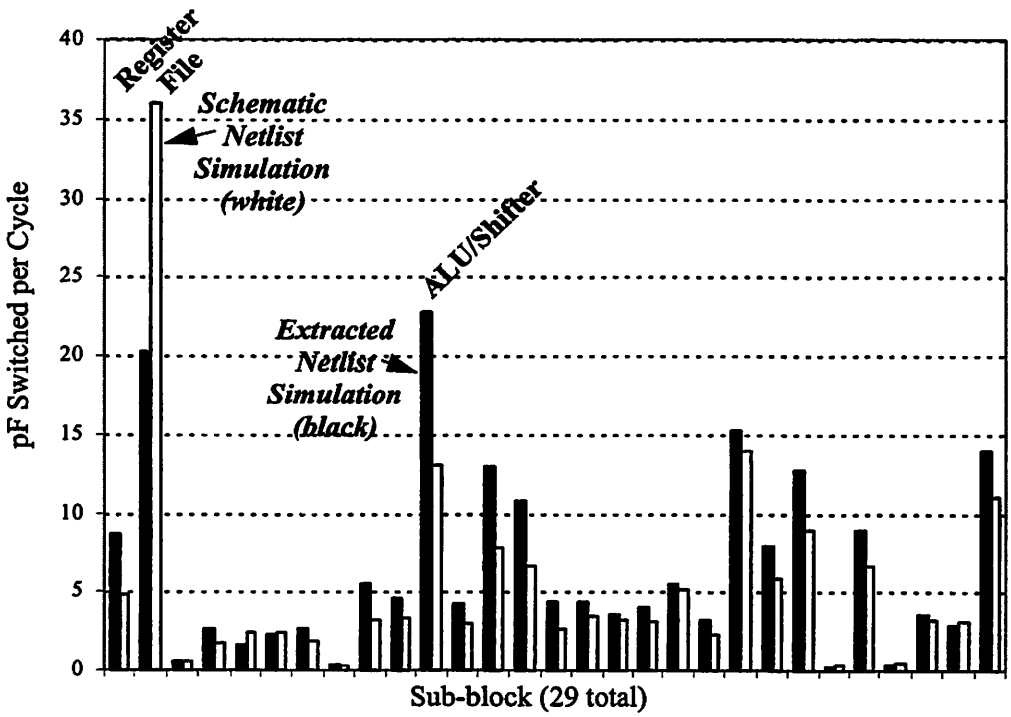


FIGURE 7.10 : ARM8 Capacitance Comparison of Schematic vs. Extracted Simulation.

In summary, the bulk of the design for energy optimization occurred while designing the schematics for the various sub-blocks, which could be simulated individually providing fast feedback on the measured capacitance per cycle. For those sub-blocks that were below 3% of the budgeted capacitance (4.5pF), which were a majority (18 of 29 sub-blocks), only obvious energy optimizations were made and the schematics were quickly mapped to layout. This allowed more time to be spent on the nine remaining blocks to be carefully optimized for energy efficiency, making the best use of the design effort and yielding an overall energy-efficient processor core implementation.

7. 2. 2. 6 Core Energy Breakdown

A breakdown of the processor core energy consumption is shown in Figure 7.11. The numbers were generated from a 25,000 cycle simulation of typical machine code on the extracted layout of the entire core. To ensure that this was a reasonable

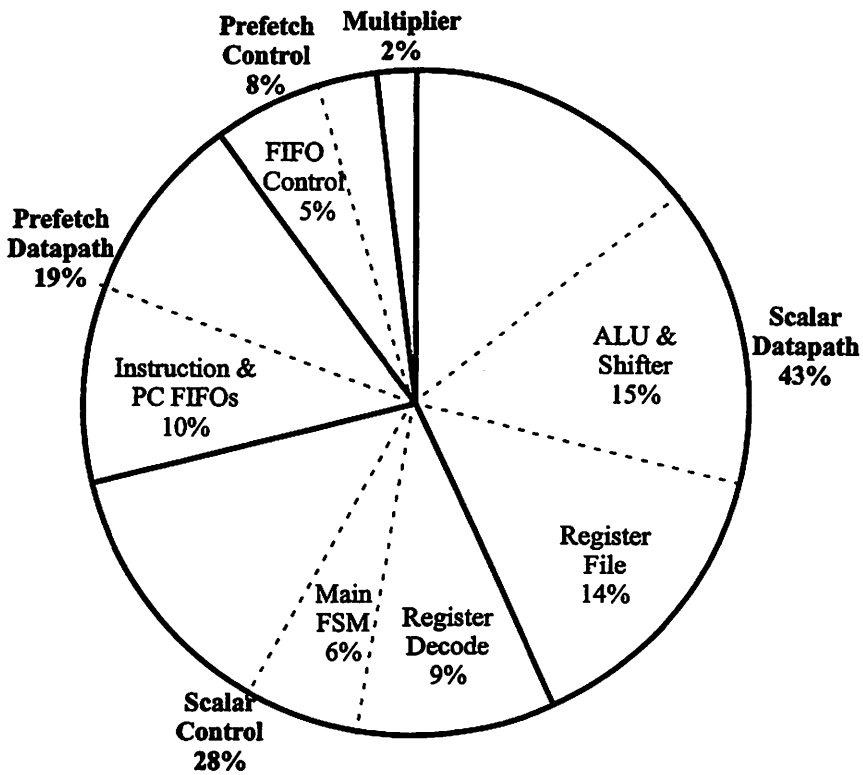


FIGURE 7.11 : ARM8 Energy Breakdown (Full Core Simulation).

simulation, it was observed that the energy consumption is within 10% of the final value in less than 10% of the simulation time. The simulation takes into account processor core stalls due to memory system latency since the input vectors were generated from a full system simulation. However, during this simulation, the processor core is never put into sleep mode, and requires an average effective switched capacitance of 187 pF/cycle while the processor system is active.

The breakdown demonstrates that the scalar core consumes 71% of the total energy, split 60-40% between the custom-layout datapath and the fully synthesized control logic. The prefetch unit consumes 27% of the total energy, split 70-30% between the datapath and control, while the multiplier consumes only 2% of the total energy for typical machine code. Among all sub-blocks, only six of them contribute 59% of the total capacitance/cycle, and it was on these six sub-blocks that a significant fraction of the design time was spent.

The register file, ALU, and shifter combined contribute 54 pF/cycle, validating the assumption in Section 7. 2. 2. 5, which estimated the capacitance at 50 pF/cycle, and subsequently utilized to derive the capacitance budget for the entire core. These three blocks' capacitance/cycle is only 29% of the total (40% of the scalar datapath/ control) which validates the initial assumption that these three blocks would contribute one-third of the total core capacitance budget of 150 pF/cycle.

7.2.3 Cache

For scalar processor cores, the cache typically dominates the total microprocessor energy consumption. However, since there was complete freedom in the design of the cache, this implementation was heavily energy-optimized, yielding a very energy-efficient cache that consumes only about one-half of the core energy consumption. The only constraint on the cache was that it should be unified, and support two memory reads per cycle to accommodate the ARM8 memory interface. The

size of the cache was chosen to be 16kB to maximize system energy efficiency (Section 5.3.1), and was solely limited by microprocessor die size.

The cache characteristics and policies were optimized for energy-efficiency as described in Section 5.3, and summarized below:

- *Associativity*: 32-way. Each 1kB block has a 32 x 23 bit CAM for the tag lookup.
- *Line Size*: 32-bytes. There are 32 lines per 1kB block.
- *Write Policy*: Write-back. Main memory is updated only when a dirty cache line is replaced in the cache.
- *Write Miss Policy*: No write allocate. Write misses are sent directly to external memory, and are not placed within the cache.
- *Replacement Policy*: Round-robin. Successive lines in the 1kB block are chosen for replacement upon a cache miss. A line will not be replaced until the 33rd cache miss to a particular block.
- *Double Reads*: Two words may be returned in a single cycle if the address LSB is 0. If two words are requested and the LSB is 1, only one word is returned. After the odd-address read, the ARM8 prefetch unit will become even-address aligned allowing subsequent double reads.
- *Instruction Buffer*: Each 1kB block has an implicit instruction buffer, though only one is active at a time. Consecutive instructions that do not cross a cache-line boundary can be made without activating the CAM, reducing energy consumption by 50%.

Figure 7.12 shows the how the 32-bit address space is utilized. The cache is physically addressed, eliminating the need for a translation look-aside buffer (TLB).

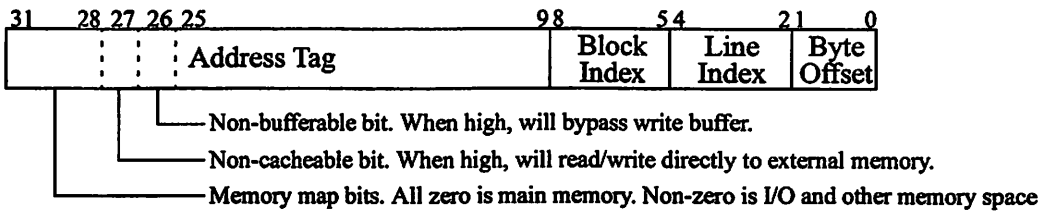


FIGURE 7.12 : Address Space Breakdown.

For embedded applications, a TLB is not particularly useful since embedded systems generally do not have a larger, secondary storage unit (e.g. disk drive) which requires a TLB to map it onto the smaller physical memory. Since a TLB also provides separate address spaces to prevent memory conflicts, the lack of one in this system forces the operating system and/or programmer to ensure no memory conflicts exist.

7. 2. 3. 1 Cache Memory Array

As a compromise between energy consumption and silicon area (as described in Section 5.3.2) the basic block size was set to 1kB, and replicated 16 times to form the cache memory array as shown in Figure 7.13. Due to the large size of the cache, which fills approximately 60% of the chip, careful attention had to be paid to routing of the busses, control signals, and power lines.

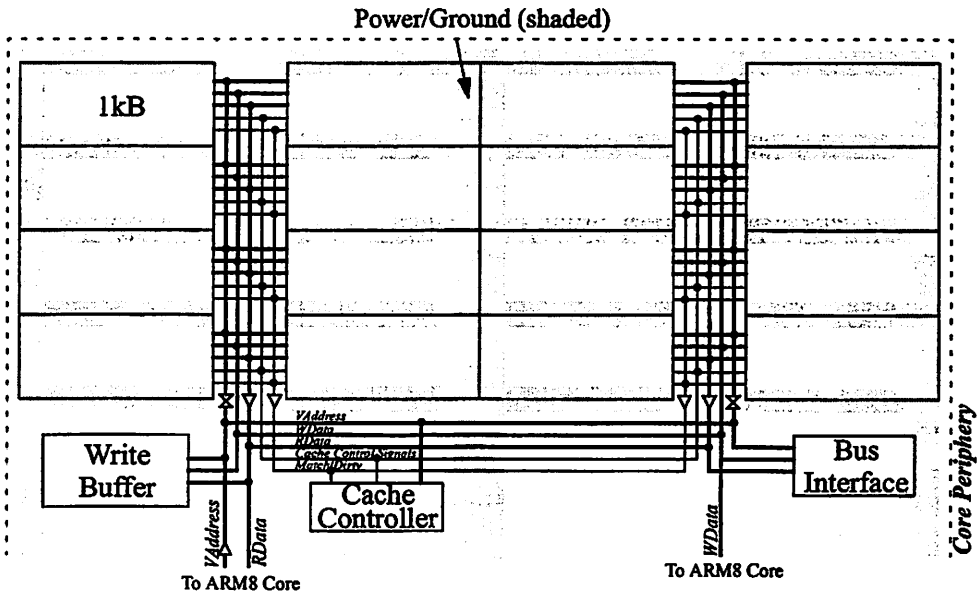


FIGURE 7.13 : Cache System Floorplan

The shaded areas indicate where power and ground are routed, which use *Metal2 & Metal3. Metal1* is used to connect up the bypass capacitance sitting under the power routes, which totals 11.7nF for the entire array. The left, right, and top sides of the cache abut the pad ring, providing low impedance from the power pins distributed around the periphery to the entire cache memory.

The switching activity of the three cache busses was analyzed to calculate the total effective switched capacitance per cycle, as shown in Table 7.1. The capacitance on *WData* is negligible, due to the low ratio of processor writes to reads, and the high correlation in the data being written. The capacitance on *VAddress* and *RData*, however, is significant (2.2% and 8.1%, respectively, of the total chip's capacitance/cycle budget). By inserting a bi-directional switch on *VAddress* and a uni-directional switch on *RData*, only half of the cache toggles per cycle, reducing the capacitance/cycle by 2.3pF (-0.8%) and 6.5pF (-2.2%), respectively. By buffering up the signal on either half of the cache, the capacitance that each block's outputs have to drive is reduced 60-75%, such that much smaller drivers can be used. The speed-up of driving less capacitance offsets the two-gate delays contributed by the insertion of the switches. To lighten the load on the *Match* and *Dirty* output signals, they too have unidirectional switches, reducing each block's output capacitance that it has to drive by 60%.

TABLE 7.1 Cache Bus Switching Activity and Effective Switched Capacitance.

Bus	Toggles (0→ 1) per cycle	Capacitance per Bit			Total Bus Capacitance per Cycle
		Half Cache	Global	Total	
VAddress	1.95	1200 fF	900 fF	3300 fF	6.4 pF
Wdata	0.15	600 fF	840 fF	2040 fF	0.3 pF
Rdata	6.50	1000 fF	1750 fF	3750 fF	24.4 pF

Thus, through simple high-level simulation and energy analysis, more than 3% of the total microprocessor's energy consumption was reduced with the addition of these simple switches.

7. 2. 3. 2 Cache 1kB Macro

The 1kB macro builds upon a previous energy-efficient SRAM design [burs97], which was ported from a 2-level metal process to a 3-level metal process. The additional metal layer was used to provide much better power distribution and reduce the capacitance on the bitline.

The architecture of the macro is shown in Figure 7.14. On the left side is the CAM array which contains the 23-bit address tags for the 32 cache lines. Upon a cache read, *prematch[31:0]* is precharged and the 23-bit address tag is passed into the CAM array, described in further detail in Section 6.2.2. If the *n-th* tag in the CAM array matches, the *prematch[n]* signal remains asserted while all the other bus signals are pulled low. The CAM state latches block contains the valid state bit for each of the 32 CAM tag addresses. The asserted *prematch[n]* signal is AND-ed with its corresponding valid state bit to indicate whether a valid match exists, and if so, the signal *match[n]* gets asserted as well as the global *Match* signal which is sent to the cache controller. This indicates the desired cache line is present in the block.

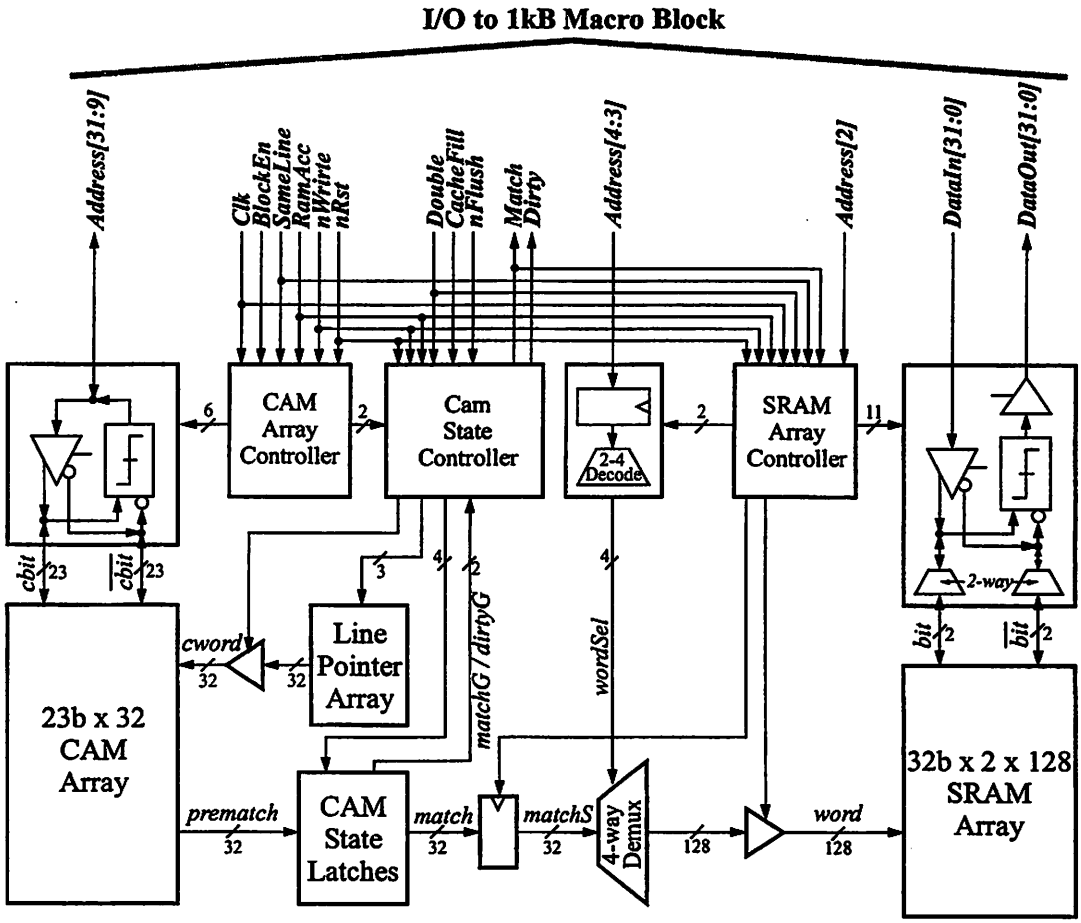


FIGURE 7.14 : Cache Memory 1kB Macro Architecture.

The *match[31:0]* bus gets latched for subsequent sequential cache reads. The *matchS[31:0]* bus is demultiplexed by *Address[4:3]* to select the desired word-pair of

the cache line, and drives the corresponding *word[m]* signal into the SRAM array. For a cache read, the two 32-bit data words corresponding to *word[m]* are read from the array, and *Address[2]* drives the column demultiplexer to select which one of the two words to place onto *DataOut[31:0]*. A cache write will take the data off *DataIn[31:0]* and write it to the desired location in the SRAM array. The schematic of the SRAM array cell, column decoder, and sense-amp is described in Section 6.2.1.

During a cache read, if the global *Match* signal remains low, indicating the cache line is not present in the block, then the cache controller looks at the global *Dirty* signal. *Dirty* is set at the end of the match operation if the next location to be replaced in the CAM has been written to in the cache and needs to be updated in main memory before replacing. The next location is determined by the line pointer array block, which is a circular chain of 32 latches, and gets rotated when a new cache line is written to the macro block. If the cache line is dirty, then the cache controller reads the address tag out of the CAM array, and then its corresponding cache line, which is then written to main memory. To place new data into the cache, the cache controller first writes the new address tag to the CAM array, and then subsequently, the eight data words corresponding to this cache line.

Because *matchS[31:0]* latches the last cache line that matched, subsequent cache accesses, which are sequential and do not wrap to the next cache line, do not need to access the CAM. Instead, the cache controller increments the cache-line index bits *Address[4:2]* appropriately, and *matchS[31:0]* drives the desired *word[m]* line to access the SRAM array.

7. 2. 3. 3 Cache Controller

The cache controller state diagram, shown in Figure 7.15, contains 30 unique states. It is the central controller for the entire cache system, driving not only the 16kB cache memory, but also routing data to the write buffer and to/from the external

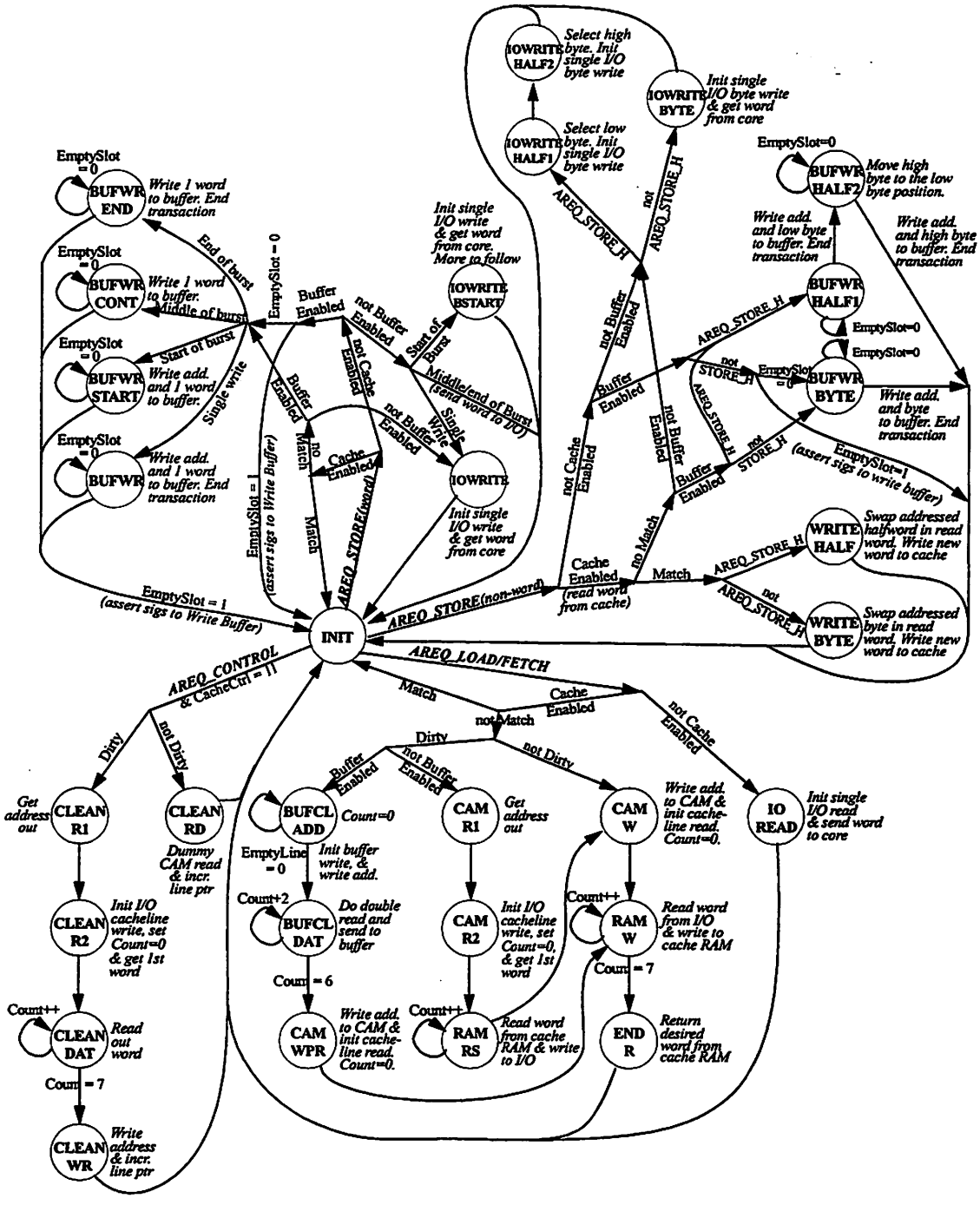


FIGURE 7.15 : Cache Controller State Diagram.

interface. The bulk of the states are required to manage the cache memory, which include writing dirty cache lines to main memory, reading in new cache lines, flushing the cache memory, and performing read-modify-write operations to support the ARM8's byte and half-word operations.

To demonstrate the timing of the cache system, a timing diagram for a double-read to the cache is shown in Figure 7.16. The ARM8's address and control signals arrive at the cache controller during the *Phi2* clock phase. The cache controller must set both the correct block enable signal and the macro block control signals by the end of the *Phi2* clock phase, so that they are stable when the CAM is accessed in the *Phi1* clock phase. The cache controller must always return a response to the ARM8 by the end of *Phi1*. If *Match* remains high, then the cache line was found in the macro block, and in the subsequent two clock phases, data is returned to the ARM8 via *RData*. If *Match* goes low, then the cache controller will lower *Confirm* (not shown) in the next

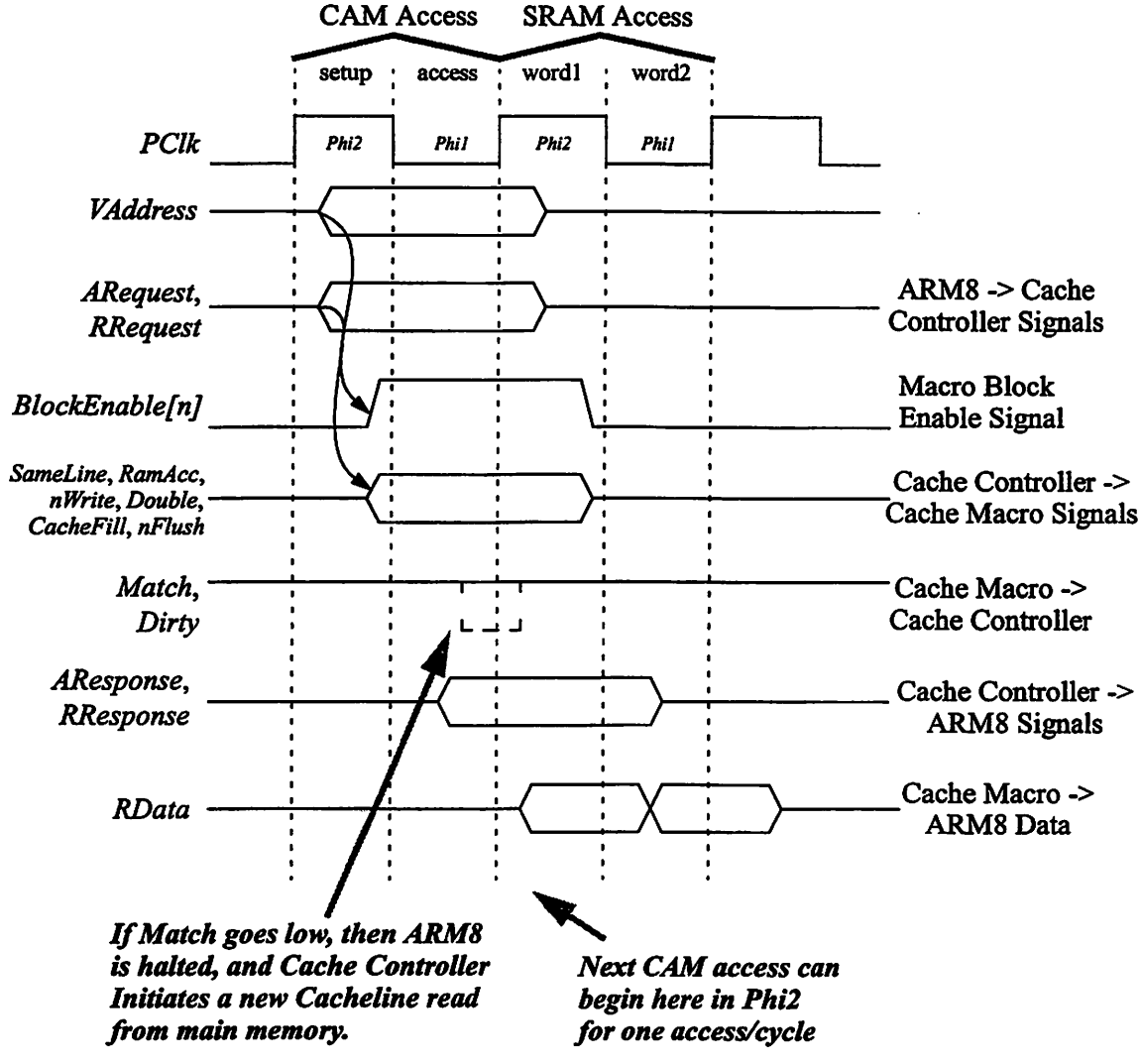


FIGURE 7.16 : Cache Memory Timing for a Cache Double-Read Hit.

Phi2, which will gate the clock to the ARM8 core while the cache system fetches the desired word. In the meantime, the cache controller loads in the desired cache line from main memory, and if *Dirty* was also low, then it writes out the old cache line back to main memory.

7. 2. 3. 4 Cache Design Optimizations

The cache controller's control signals to the write buffer, bus interface, and the ARM8 core are dependent upon whether the *Match* signal, which is output by the activated cache macro block, is high or low. The signal is not available until late in *Phi1*, and created a critical path for generating these control signals, which must be available at the beginning of the next *Phi2*. All possible cache accesses were analyzed with the C simulator and categorized as either common cases or rare cases as shown in Table 7.2. To reduce the critical path, an extra cycle delay was added to the state machine for the rare cases, in order to reduce the loading on *Match* and speed up the critical path.

TABLE 7.2 Categorization of Cache Access Types.

Common cases requiring optimization:
1. Cache read hit (single & burst)
2. Non-cacheable single read
3. Cache read miss (single & burst)
4. Cache write hit (single & burst)
5. Non-cacheable/bufferable write (single & burst)
6. Non-cacheable/non-bufferable write (single & burst)

Rare cases which could be slowed down:
1. Non-cacheable burst reads
2. Cacheable/ bufferable write miss (single & burst)
3. Cacheable/ non-bufferable write miss (single & burst)

Byte and half-word reads are rotated by the core. Byte and half-word stores must be rotated by the memory system. Since the cache controller only consists of standard cells, the datapath logic to do this resides in the bus interface, where the *RData* and *WData* busses are readily available. However, this datapath logic is directly controlled by the cache controller. Writes to memory locations present in the cache memory require one stall cycle so that a read-modify-write can take place, as shown in

the timing diagram in Figure 7.17. The original data word is read from the cache memory, and latched from *RData* onto *RDataT2*. The byte or half-word to be written is latched off of *WData* onto *WDataT1*, then merged with the saved data on *RDataT2* and placed back onto *WData* where it can be written to the cache memory. Write misses are sent as byte writes to either the write buffer or the bus interface. Because the external SRAM and I/O can only operate on words and bytes, two cycles are required for half-word write misses, in which *WDataT1* is used for temporary storage, in order to split the half-word into two byte writes. When *Confirm* goes low, the ARM8 core is stalled for either one or two cycles depending upon whether it is a byte or half-word write.

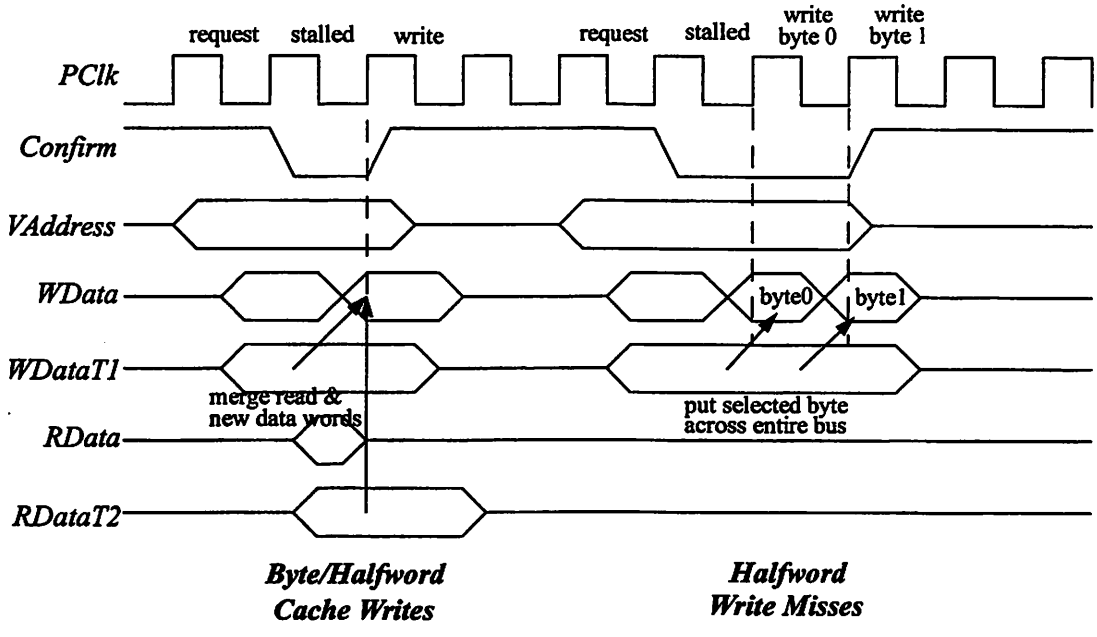


FIGURE 7.17 : Timing for Non-word Writes to the Cache and Main Memory.

7. 2. 3. 5 Cache Energy Breakdown

As shown in Figure 7.18, half of the energy consumed by the cache occurs in the SRAM component of the cache memory. For a single cache access in which the CAM is activated, the CAM consumes 60% of the energy consumed by the SRAM. But with the virtual instructions buffers, activation of the CAM is suppressed for sequential instruction fetches, reducing its average energy consumption to 20% of that of the SRAM. The busses consume 25% of the total cache energy, and includes the energy

consumed by the address buffer. Finally, the cache controller consumes 15% of the total cache energy. On a cycle-by-cycle comparison, the cache consumes 63% of that consumed by the ARM8 core.

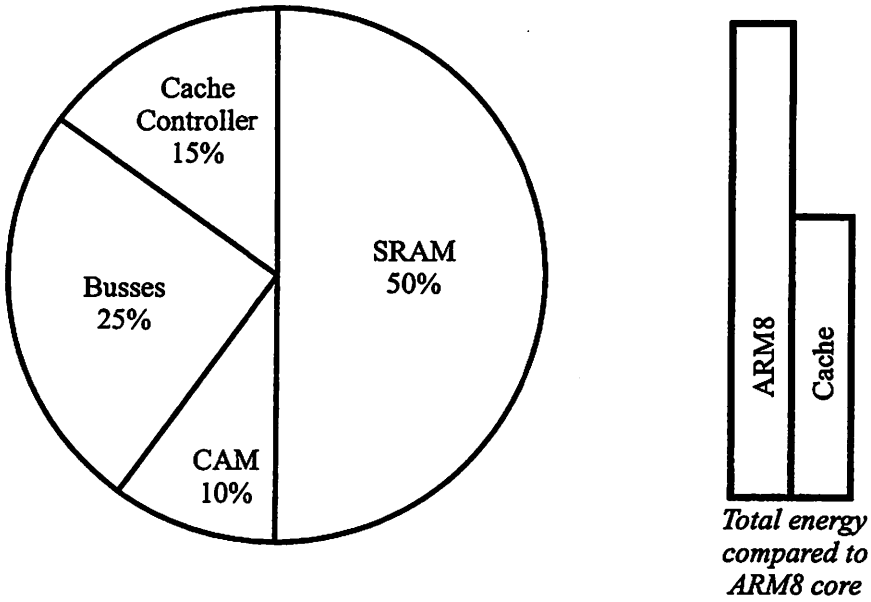


FIGURE 7.18 : Cache Energy Breakdown.

7.2.4 Write Buffer

Since the external bus multiplexes address and data onto the same bus, the write buffer stores both the address and data in a single register file, as shown in Figure 7.19. The multiplexed architecture allows either one cache line and one store, or up to six single-word stores. In addition, the buffer can store a variable-number of multiple words per single address for the Store Multiple (STM) instruction. If the STM words cross over a cache-line boundary, the beginning address of the second cache line is

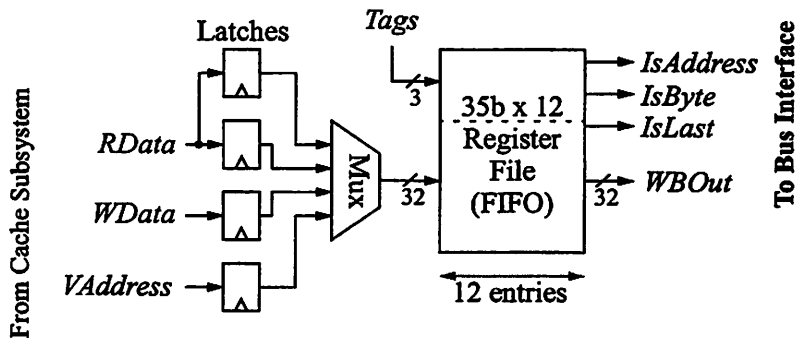


FIGURE 7.19 : Write Buffer Architecture.

placed into the buffer to align the external memory access on a cache-line boundary. This is required to ensure that the store does not cross over multiple external SRAMs, which the bus interface cannot support. Simulation demonstrated that given the system architecture, any more than twelve buffer entries yields negligible performance improvement.

The register file is 35 bits wide. The additional three bits are tags used to indicate if the entry is an address (*IsAddress*), if it is the last data word (*IsLast*) or if it is a byte-wide store (*IsByte*). The three busses of the cache subsystem (*VAddress/ Wdata/Rdata*) are latched and multiplexed going into the register file. The latches are required to provide enough setup and hold time for the register file, with two latches required for *RData* because when reading out a line from the cache memory, two words are returned per cycle. The address/data words are placed onto *WBOut* and sent to the bus interface, under its control.

An out-of-order write buffer requires hardware to compare the address of a pending read to all the addresses stored in the write buffer to ensure memory consistency. The buffer control was significantly simplified by enforcing all external memory accesses to be in order; before any external read request, the write buffer is first flushed out. The exception to this rule is cache-line reloads, which are guaranteed not to have the same address between the cache line being written out and the new line being read in. Providing this exception reduces the latency to complete a cache-line reload by a factor of two.

The input to the write buffer is controlled by the cache controller via four signals, shown at the bottom of Figure 7.20. The *LoadWord* signal is utilized to enable the write buffer, while the other three signals are decoded to determine which bus to latch (*LoadDirect*), which word is the address (*LoadAdd*), and which data word is the last one (*LoadLast*). The timing on the input busses was dictated by the ARM8 memory interface, and the timing of the cache memory.

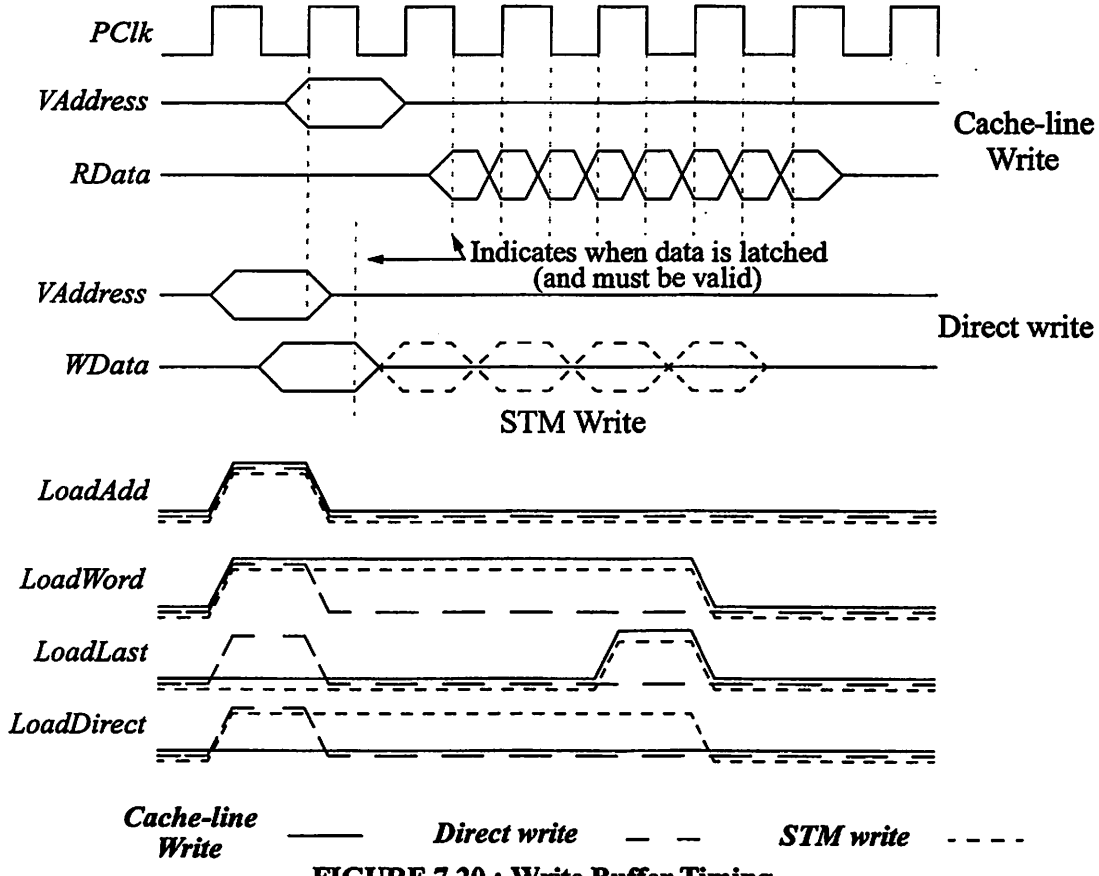


FIGURE 7.20 : Write Buffer Timing.

7. 2. 4. 1 Energy Consumption

For single writes (STR), the effective switched capacitance is 63 pF/word (from simulation on extracted layout), and 26 pF/word for multi-word stores (STM). The only instructions which use the write buffer are non-cacheable stores and stores that take a cache miss. Since stores are approximately 10% of the instruction mix, the write buffer contributes 1.3 pF/cycle, on average. Read cache misses may also enable the write buffer, but only if the cache line is dirty, and switch 110 pF/cacheline. However, this condition occurs well under 1% of the time, so the overall contribution is less than 1 pF/cycle, on average. Thus, the write buffer consumes less than 1% of the total processor chip energy consumption.

7.2.5 Bus Interface

The primary responsibility of the bus interface is to connect the processor

clock (*PClk*) and bus clock (*MClk*) domains. The bus interface also includes the components to enable byte and halfword writes, as shown in Figure 7.21. Reads from external memory typically take many cycles to complete, and will stall the ARM8 core and/or cache system until the external access has completed. Since the prototype processor is an in-order machine, there is no reason to provide buffering for reads to allow the core to continue operating, since it must wait for the pending word to continue. With a separate write buffer, there is no need to provide additional buffering within the bus interface, such that the bus interface complexity is reduced to a four-to-one multiplexer, four registers, and enabled buffers to drive the cache-system busses.

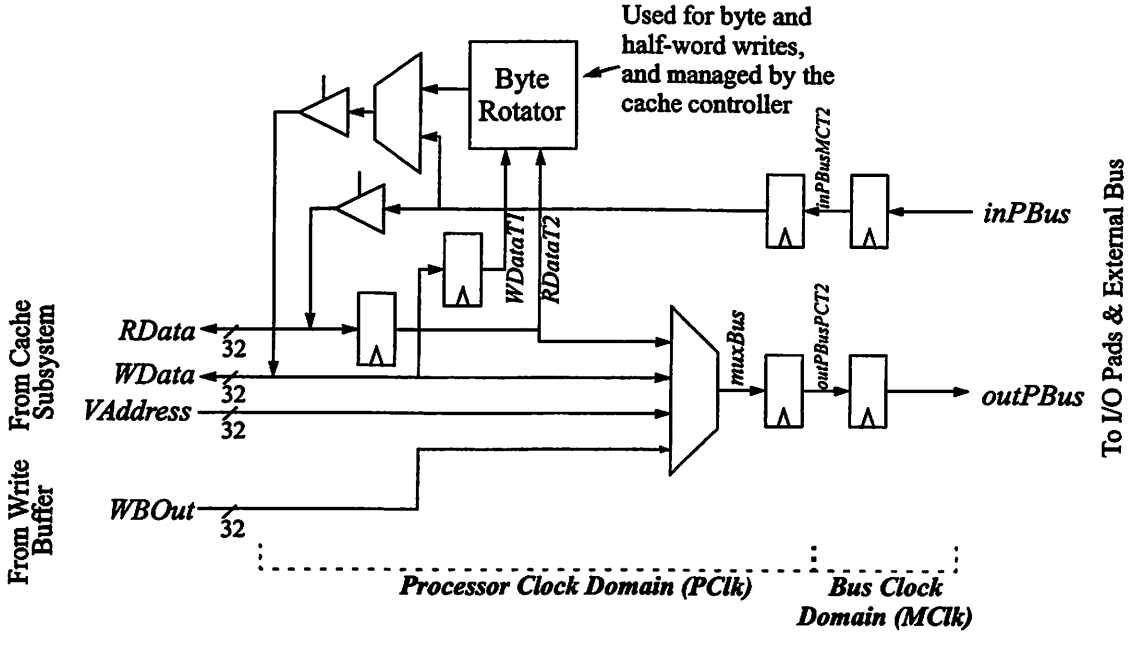


FIGURE 7.21 : Bus Interface Architecture.

The bus interface talks to the cache controller, and the core via the controller, on one side of the interface running at the processor clock speed (*PClk*). The other side of the interface communicates with external memory at the processor bus clock speed (*MClk*). The *MClk* speed is programmable via the system coprocessor, and can operate at a 2x, 4x, or 8x multiple of *PClk*. Initially there was a 1x option, but the additional hardware to support this was not warranted given the marginal performance improvement achieved.

The state machine controlling the bus interface is relatively simple, as shown in Figure 7.22. If the state machine is idling, it services the write buffer if it is not empty. Otherwise, it services I/O requests from the cache controller. Maintaining this order ensures memory consistency. Otherwise, read requests from the cache controller would have to be matched against pending writes in the write buffer, at the expense of significant hardware complexity. Simulation demonstrated little performance degradation by enforcing this order to eliminate the extra hardware. The only instance when the cache controller takes priority over the write buffer is for a dirty cache-line load, in which it is guaranteed that the outgoing cache line is not the same memory location as the incoming cache line.

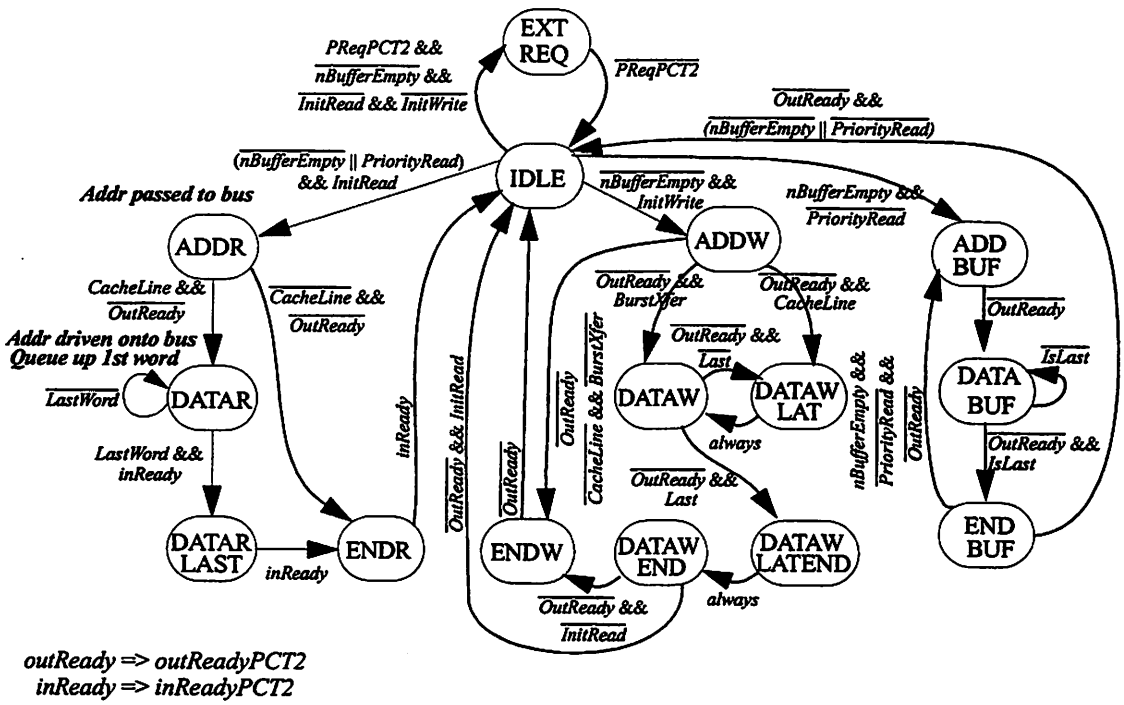


FIGURE 7.22 : Bus Interface State Machine.

All bus transactions must complete before moving onto a new transaction. The state machine operates at *PClk* speed, and sends signals to the bus-side logic via a simple handshake scheme that is independent of the phase difference between *PClk* and *MClk*. If the *PReq* signal is asserted by the I/O chip, indicating a pending direct memory access (DMA) request, the state machine hands off control of the bus after completing

all outstanding bus requests. The I/O chip can then directly access main memory. The processor core can continue to run, but if it attempts to access the bus, or attempts to write to a full write buffer, then the core will stall until the I/O chip releases ownership of the processor bus.

7. 2. 5. 1 Clock Interfacing

MClk is derived from *PClk* using a selectable frequency divider consisting of three flip-flops and a multiplexer, which introduces some phase shift. The *MClk* signal actually used is a buffered version of the signal off the external clock pad, which ensures that the processor chip, memory chips, and I/O chip all operate with an *MClk* that has minimum relative phase shift between the chips. This improves the robustness of the signal timing on the external processor bus. This buffered *MClk* signal introduces additional phase shift with respect to *PClk*, but a self-timed handshake scheme allows proper operation of the bus interface independent of this phase shift, as well as V_{DD}.

Core->Bus: The state machine changes state in *Phi2* of *PClk*, and all signals going to the external bus are derived from the machine state and other *Phi2* signals. On the falling edge of *PClk*, the control signals are latched, and the signal *outReady* is asserted via an $\overline{\text{RS}}$ latch, with some delay. The *outReady* signal is then latched when *MClk* is low, generating *outReadyMCT1*, to ensure a stable signal when *MClk* is high. Upon the next rising edge of *MClk*, if *outReadyMCT1* is high, the output data is latched, placed on *outPBus*, and sent directly to the processor bus pads. At the same time, the bus-side logic drives *outReady* low via the same $\overline{\text{RS}}$ latch, and *outReady* is latched on the rising edge of *PClk* to generate the signal *outReadyPCT2*. This signal is used by the state machine to either wait, or pass new data to the processor bus. When *outReady* and *MClk* are coincident in time, if the bus-side logic detects it is high, the additional delay generating *outReady* will ensure the processor-side data is valid. Otherwise, *outReady* will not be detected until the subsequent rising edge of *MClk*. The signal *outReadyPCT2* can stay high for up to one *PClk* cycle after the rising edge of *MClk*, but since the

lowest clock ratio is 2x, the core-side logic will set up the next data element in the second cycle, and be ready for the next slot on the processor bus. Hence, the state machine at all times will be able to maintain maximum throughput on the external processor bus. The timing for when the edges are coincident are demonstrated in Figure 7.23.

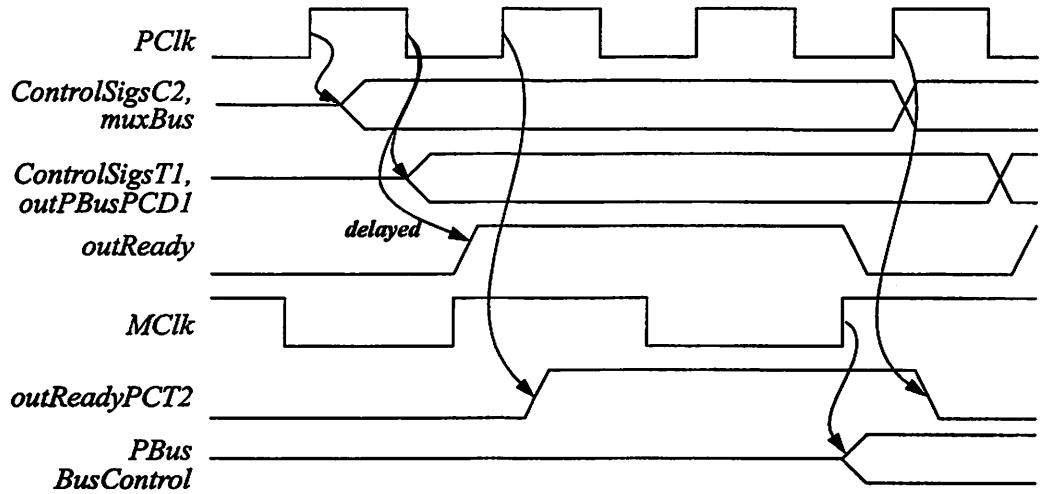


FIGURE 7.23 : Core-side to Bus-side Timing.

Bus->Core: This case is similar to the previous one, in which all outgoing processor bus requests get latched on the falling edge of *PClk*, and latched on the next rising edge of *MClk* when *outReadyMCT1* is high. There is an additional signal that is sent to tell the bus-side logic to latch the external processor bus on the falling edge of *MClk*. When this occurs, the signal *inReady* gets asserted via another $\overline{RS}$ latch, which is latched on the next rising edge of *PClk* to generate the signal *inReadyPCT2*. Again, additional delay is placed on the *inReady* signal to give the latched data enough time to settle. The signal *inreadyPCT2* is then used by the state machine to latch the data taken of the *inPBus* bus, and send it off to the cache or processor core.

It is possible, if the *outReady* signal is exactly coincident with the rising edge of *MClk*, for it to be detected some cycles, and not for others. This has the potential of placing a bubble on the bus if a missed cycle follows a caught cycle, leading to an invalid bus operation and possible system failure. To prevent this, an additional signal

can be utilized to shift *MClk* an additional eight gate delays. This will eliminate the coincidence, thereby eliminating the coincidence. Fortunately, this feature was not required for correct operation of the prototype processor system.

7. 2. 5. 2 Energy Consumption

Due to its simplicity and infrequent use, the bus interface has very little energy consumption, on the order of 1-2% of the total processor energy consumption. However, this does not include the bus drivers located in the chips pads, which consume considerably more energy due to the large capacitance on the external bus, on the order of 5-10% of the total system energy consumption.

7.2.6 System Coprocessor

The system coprocessor is a standard component of an ARM-based microprocessor system, and is commonly found in some form in most other microprocessors as well. It is responsible for system-level control functionality which is independent of the processor core, as well as configuring the specifics of the processor core.

The coprocessor operates lock-step with the ARM8 pipeline, but with a half-cycle delay as shown in Figure 7.24. For example, the coprocessor's *CDecode* stage starts on the rising edge of *PClk* in the middle of the core's *Decode* stage. All instruction fetching is performed by the core, so the coprocessor has no fetch stage. There is also no memory stage since the coprocessor cannot directly access memory.

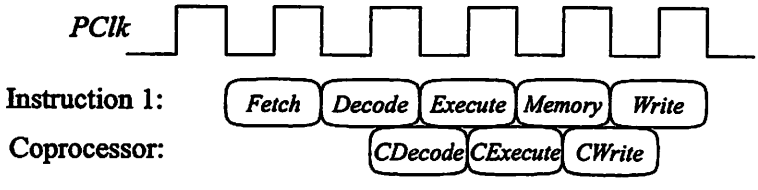


FIGURE 7.24 : ARM8 Coprocessor Pipeline.

The ARM8 interfaces to the coprocessor via three busses. The pre-decoded coprocessor instruction is placed onto the *CInstruct* bus in the first half of the

processor's *Decode* stage so that it is available on the rising edge of the clock. On this edge, the coprocessor enters the *CDecode* stage in which the coprocessor instruction is decoded. The coprocessor then proceeds with its three-stage pipeline, as shown in Figure 7.24. Reads from the coprocessor to the ARM8 are performed via the *CData* bus which is driven during the *CExecute* stage so that data can be written to the ARM8's register file in the second-half of the *Memory* stage. Writes to the coprocessor are done by placing the data value on *VAddress* during the *Memory* stage, which makes the data available to be written into the coprocessor's register file during the *CWrite* stage.

The system coprocessor is comprised of various counters and registers containing special state variables. While the block appears logically like a register file, it could not be implemented as such due to the heterogeneity of the registers; some are read-only counters while other are read-write registers. Also, on many of the logical read-write registers, many of the bits are hard-coded to zero. The implementation used a shared bus architecture, with separate input and output ports. Table 7.3 lists all the registers of the system coprocessor, which is logically organized as three separate coprocessors.

TABLE 7.3 System Coprocessor Register Summary

Reg#	Coprocessor 13	Coprocessor 14	Coprocessor 15
0	Access Cycle Count (RO)	Real Time Counter Low (RO)	MMU ID (RO)
1	Idle Cycle Count (RO)	Real Time Counter High (RO)	System Control
2	Sleep Cycle Count (RO)	Timer Interrupt	*not used*
3	Wait Cycles (RO)	Interrupt Suspend	
4	Hit Count (RO)	Internal Dynamic Clock Speed	
5	Cached Miss Count (RO)	External Pin Control	
6	Cache Writeback Count (RO)	Hw. Control Tweaks	
7	Uncached Access Count (RO)	Instruction Count (RO)	Cache Operations (WO)

7. 2. 6. 1 Coprocessor 13

This logical coprocessor only consists of read-only counters. Four of the counters (register 0-3) are used to monitor processor operation. One counter records

cycles that the processor core is making a memory request (Access); another tracks when the core is active, but has no memory request (Idle); a third tracks when the processor is asleep (Sleep); and the last one tracks when the core is stalled waiting on the external bus to complete a transaction (Wait). Another four counters (register 4-7) monitor cache operation by recording the number of cache accesses that are hits, misses, dirty cache-line writebacks, and uncached accesses. These eight counters can be utilized by the operating system to adjust processor performance depending upon how the processor is being utilized. For example, if the processor spends a significant amount of time stalled, then processor speed can be reduced because the performance bottleneck is in accessing I/O data.

7. 2. 6. 2 Coprocessor 14

Registers 1 and 0 are read-only, and form a 64-bit real-time counter whose value is in microseconds. When writing to register 2, any pending timer interrupt is cleared, and a new timer value is set, also in microseconds. When reading this register, the next timer event is returned. The processor enters sleep mode when register 3 is written to. The processor will remain idle until the next interrupt occurs, either due to an external event or due to the timer. Reading register 3 returns the current state of the interrupt lines as indicated in Table 7.4. The *EnIRQ* bit as specified indicates a pending *IRQ* request from an external source, while the *nTIQ* line indicates a pending timer interrupt from the internal timer. The *EnIRQ* and *nTIQ* lines are merged into a single signal, *nIRQ*, which is then sent to the ARM8 core.

TABLE 7.4 Interrupt Information Bitmap. (CP14R3)

31-10	9	8	7	6	5-0
x	*nTIQ*	*EnIRQ*	*nIRQ*	*nFIQ*	x

A write to register 5 sets two target internal dynamic clock speeds: one for normal operation, and one for interrupts as shown in Table 7.5. The special interrupt clock speed can be enabled/disabled with a separate control bit for both *IRQ* and *FIQ* in

C15R1. Upon writing to this register with no pending interrupts, the desired clock value is sent to the regulation system via the regulator interface (Section 7. 2. 6. 4). When either an *FIQ* or *IRQ* arrives (and the corresponding mask bit is enabled in C15R1), the interrupt clock speed is sent to the regulation system. Also, upon de-assertion of the interrupt, the normal clock rate is sent to the regulation system. Reading this register returns the current sampled processor speed, which is not necessarily the same value written. This allows the operating system to get feedback from the voltage converter loop to ensure that it is delivering the target frequency.

TABLE 7.5 Clock Speed Write Bitmap. (CP14R4)

31-15	14-8	7	6-0
ignored	interrupt clock speed	ignored	normal clock speed

The lower three bits of register 5 controls the state of four external pins. There is no other effect of writing to this register, and thus it is the recommended register to use when NULL coprocessor write operations are required. A read from this register returns the last value written. Register 6 controls both the external bus clock ratio (bits 6:5), and the fine-tuning for the VCO (bits 4:0), which is described further in Section 7.2.7. The bus clock ratio can be set to 2x (10 or 11), 4x (01), or 8x (00). Register 7 is a read-only register which maintains a count of the number of instructions executed since processor start-up.

7. 2. 6. 3 Coprocessor 15

Coprocessor 15 contains standard register definitions in ARM implementations [arm96a]. However, only those registers that pertain to the prototype system architecture were included; the registers that control a translation look-aside buffer (TLB) were not implemented due to the lack of a TLB in the prototype system. Register 0 is read-only and always returns the hexadecimal value 0x42018110, which specifies the implementor, 0x42 ('B' for Berkeley), the architecture version, 0x01, the part number, 0x811, and the revision, 0x0.

Register 1 is the system configuration register, whose 15 standard bit mappings are described in Table 7.6. Those bits which apply to the prototype are in ***bold***. The *C*, and *W* bits effect the function of the cache memory system. The *B* and *Z* bits are fed back into the processor core to alter core functionality. The *A* bit is sent to the cache controller to alter response to non-aligned memory accesses. All others bits are read as 0 or 1, and are unalterable.

TABLE 7.6 System Configuration Register Bitmap. (CP15R1)

	31-15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	not used	***IE***	***FE***	*I*	***Z***	*F*	*R*	*S*	***B***	*L*	*D*	*P*	***W***	***C***	***A***	*M*
Initial Value	0.....0	**0**	**0**	0	**0**	0	0	0	**0**	0	1	1	**0**	**0**	**0**	1
Description	***A***: Alignment Fault Enable ***C***: Cache Enable ***W***: Write-buffer Enable ***B***: Big Endian Select (else Little Endian) ***Z***: Branch Prediction Enable ***FE***: Enable different clock speed for FIQ (set with C14R4). ***IE***: Enable different clock speed for IRQ (set with C14R4).															

Writing to register 7 will flush, or clean, cache blocks. The Flush operation will invalidate the entire cache. The Clean operation writes out data at the specified address if it is dirty. The entire cache can be cleaned by stepping through all 512 cachelines. These operations also require subsequent writes to the NULL coprocessor register (CP14R5) to work properly with the cache system. The code sequences are shown in Table 7.7.

TABLE 7.7 Cache Control Operations. (CP15R7)

Function	opcode_2 value	CRm value	Data	Instruction
Flush ID cache(s)	000	0111	SBZ	MCR p15, 0, *X*, c7, c7, 0 MCR p14, 0, *X*, c5, *X*
Clean ID single entry	001	1011	VA	MCR p15, 0, Rd, c7, c11, 1 MCR p14, 0, Rd, c5, *X*

(SBZ = Should Be Zero, VA = Virtual Address, *X* = don't care)

7. 2. 6. 4 Regulator Interface

In the prototype processor, the system coprocessor is also responsible for

interfacing with the separate regulator chip, when the conditions for changing the processor frequency occur (Section 7. 2. 6. 2). The interface is synchronized to the regulator chip with the 4 MHz clock signal, and transmits the new seven-bit digital frequency value serially, in order to reduce the pins required.

Once the regulator has received the new frequency value, and begins adjusting V_{DD} and the clock frequency accordingly, further frequency change requests must be blocked until the regulator has reached steady-state. However, it is not necessary to do this on the processor. On the DVS chip, new request are denied as long as the internal *Track* signal is high, which indicates that the converter is currently changing V_{DD}, so there is no need for flow control from the converter chip back to the processor. Hence, the only time new requests will be blocked by the interface is when there is a currently pending transaction being serially transmitted.

7. 2. 6. 5 Energy Consumption

There are three different processor operating conditions to consider when analyzing this block's energy consumption: Active (processor is active), Wait (processor is stalled on a memory access), and Sleep. The first is not critical, since the energy consumed by the processor core will dwarf that consumed by the coprocessor. The second is not critical either, due to energy consumption in the cache subsystem which dominates any energy consumed by the coprocessor, and because the energy will be the same between the Sleep and Wait modes. The Sleep mode is most critical since this is the lowest energy mode, with energy consumption dominated by the coprocessor and the global clock distribution, as shown in Table 7.8.

TABLE 7.8 Estimated Processor Capacitance/cycle by Operating Condition.

Mode	Coprocessor	ARM8	Cache System
Active	6.3 pF	200 pF	125 pF
Wait	3 pF	4 pF	40 pF
Sleep		4 pF	3 pF

Thus, the system coprocessor's circuits which are always active (e.g. real-time counters, interrupt controller, etc.) were optimized to minimize their energy consumption, which was reduced to 30% of the total energy consumed by the processor while in Sleep mode.

7.2.7 VCO

To accommodate process variation over the die, as well as simulation error, the oscillator was designed to be programmable from 50% to 150% of nominal frequency with five bits of control. The frequency control is designed to be glitch-free so that it can be programmed via software through a register in the coprocessor (CP14R6).

The basic oscillator architecture, shown in Figure 7.25, consists of five binary-weighted delay blocks, plus a return path to close the loop. Each of the delay blocks has both a slow and fast path which is selected by the *ctrl[n]* signal. A new value for this signal may be loaded when the *trig1* signal transitions low-to-high. By ensuring that the pass gates in the basic block have switched by the time *trig2* transitions low-to-high, the oscillator will change frequency glitch-free. At system start-up, the VCO operates in its slowest mode (e.g. highest voltage for a fixed frequency) to ensure proper operation. In the initial boot sequence, the operating system can measure how fast the

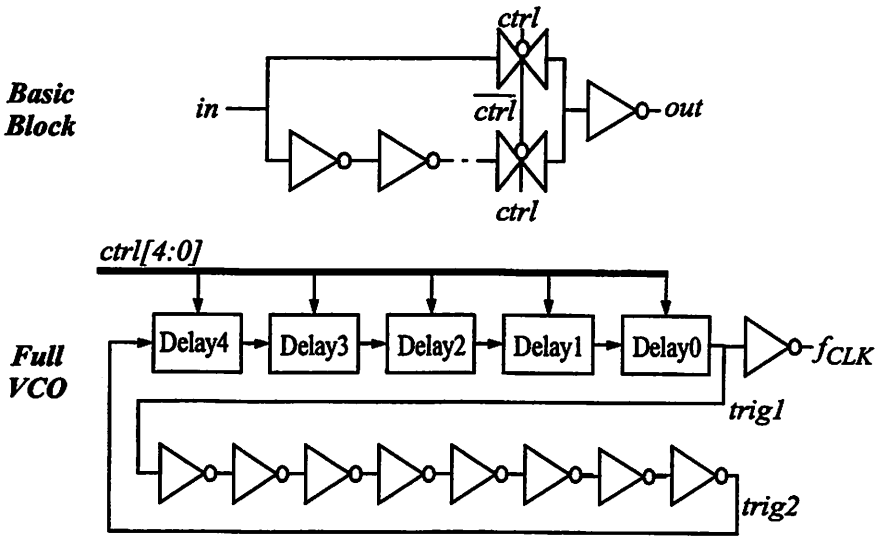


FIGURE 7.25 : VCO Architecture.

VCO can be operated at, and set it accordingly.

The hardware was stepped from 5 MHz to 80 MHz in 5 MHz increments, and at each step, the ring oscillator's control bits were decreased until processor failure. Decreasing the control bits had the effect of decreasing supply voltage, since the converter loop maintains constant clock frequency. The minimum control setting to prevent processor failure was exactly the setting for nominal frequency at all frequency values, with the exception at 5 MHz, at which speed the control could be decreased by one LSB from nominal. This demonstrates that the critical paths of a CMOS processor do track extremely well over a wide range of voltage.

7.2.8 Packaging and Chip-Level Design Issues

The microprocessor die was placed into a 132-pin QFP package. There are 77 signal pins, with 56 pins required for the processor system bus including 17 pre-decoded chip-enable signals for the memory chips (*CE[15:0]*) and the interface chip (*IOCE*). Thus, no external decoding circuitry is required, as the processor can be internally configured for 1-16 32kB, 64kB, or 128kB memory chips. Additional signal pins include four pins for the regulator chip interface, two pins for the external interrupt lines from the I/O sub-system, one pin for an initial reset by the regulator, one pin for an external reset signal, and one pin for the reference clock signal used for the internal counters. Twelve more pins are used to provide debug support.

There are 55 power pins, with 28 for ground, 16 for the variable processor core voltage (V_{DD}), ten for the variable I/O circuit voltage (V_{DDIO}), and one for the battery voltage (V_{BAT}). Although separate ground lines for the core and I/O circuits would have been preferable, in order to isolate the core from the noisy I/O circuits, the low-impedance substrate in this process makes this unfeasible. The battery voltage is strictly used to provide electro-static discharge (ESD) protection on those input pins whose signal level is V_{BAT}.

The complete processor chip pad breakdown is given below.

TABLE 7.9 Processor Chip Pad Breakdown (132-pin QFP package).

Signal	i/o	Pads	Supply	Description	Pin Number(s)	I_{MAX}
gnd!		28		Single ground	1,7,11,16,19,23,28,31,39 42,47,52,57,64,68,72,77 81,87,91,98,102106,110,115, 118,125,129	
V_{DD}		16		Core voltage (1.2-3.8V)	9,21,33,37,49,60,70,74, 79,89,96,100,108,120, 127,131	
V_{DDIO}		10		I/O voltage (1.2-3.8V)	4,14,25,35,45,84,94,104,113, 122	
V_{BAT}		1		High-V ESD (4V)	59	
MClk	o	1	V_{DDIO}	LPARM Processor Bus	126	50 mA
PBus	i/o	32	V_{DDIO}		50,48,46,44,43,41,40,38,36, 34,32,30,29,27,26,24,22,20, 18,17,15,13,12,10,8,6,5,3,2, 132,130,128	33 mA
Write	ot	1	V_{DDIO}		124	33 mA
Byte	ot	1	V_{DDIO}		123	33 mA
Burst	ot	1	V_{DDIO}		121	33 mA
PReq	i	1	V_{DDIO}		119	
nMREQ	o	1	V_{DDIO}		116	33 mA
PWait	i	1	V_{DDIO}		117	
CE	ot	16	V_{DDIO}		85,86,88,90,92,93,95,97,99, 101,103,105,107,109,111,112	33 mA
IOCE	o	1	V_{DDIO}		114	33 mA
PClk	o	1	V_{DD}	Clock output	69	50 mA
PwrGood	i	1	V_{BAT}	Reset from converter	58	
nRst	i	1	V_{BAT}	External hard reset	61	
FIQ	i	1	$V_{3.3}$	Interrupts. Input from Xilinx (3.3V)	62	
IRQ	i	1	$V_{3.3}$		63	
RefClk	i	1	V_{BAT}	1 MHz, for timers	66	
Clk4M	i	1	V_{BAT}	4 MHz, for DVS	67	
LoadM	o	1	V_{DD}	DVS interface	71	10 mA
DataM	o	1	V_{DD}		73	10 mA
ExtClk	i	1	V_{BAT}	External clock input (for debug)	56	
EnExtClk	i	1	V_{BAT}		55	
SwCE	i	2	V_{BAT}	Debugging pins	53, 54	
ExtPins	o	4	V_{DDIO}		78, 80, 82, 83	33 mA
Confirm	o	1	V_{DDIO}		76	10 mA
Stall	o	1	V_{DDIO}		75	10 mA
EnSpec	i	1	V_{BAT}		65	
ShMClk	i	1	V_{BAT}		51	

7. 2. 8. 1 Pad Design

For debugging purposes, the I/O pads were designed to operate at a different voltage than the core, so that they could be left at fixed voltage while the internal core voltage was varied. Thus, all output & input pads support level shifting, with the exception that the four signals which connect to the regulator chip must always be at the nominal core voltage (V_{DD}). The schematic for the level-converting I/O pad is shown in Figure 7.26.

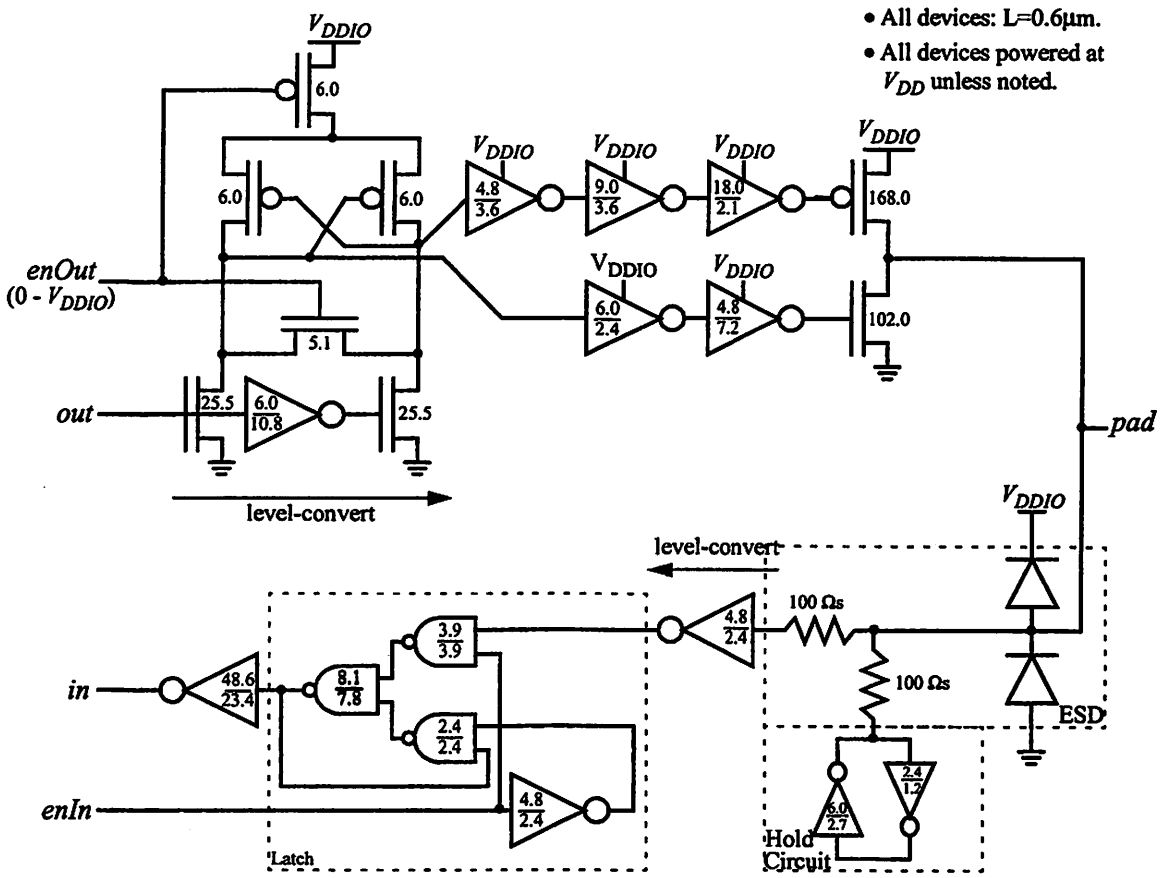


FIGURE 7.26 : Level Converting I/O Pad.

For an outgoing signal, the enabled cross-coupled loads on the complementary NMOS gates provides level conversion from V_{DD} to V_{DDIO}. The ratio of NMOS to PMOS width is dictated by the maximum possible range of voltage conversion. The level conversion was designed to operate from 1V to 3.3V, and with an effective W_P of 3µm, the required W_N was 25µm. Simulation demonstrates that this will correctly operate for V_{DD} as low as 950mV. The *enOut* signal must range from 0V to V_{DDIO}, so

the enable signal generated by the core passes through its own level-converter before driving the pads.

The target load capacitance on the output is 50pF, which is sufficient to drive ten chips (4pF each) and four inches of a PCB trace (2.5 pF/inch). Signals must be transmitted within one-half of an *MClk* cycle since they change on the rising edge and are latched on the falling edge. The target delay through the pad is one-quarter of a cycle, allowing another one-quarter cycle of margin before the signal arrives at the other chip. The target rise/fall time is also one-quarter cycle in order to reduce current draw. At 50pF and V_{DDIO} = 3.3V, this corresponds to a peak current (I_{MAX} in Table 7.9) of 33mA per signal. The size of the inverters driving the output MOSFETs was dictated by ground and power bounce concerns, and discussed further in Section 7. 2. 8. 2.

A feedback device was added to the tri-stated outputs to hold state while V_{DDIO} varies. In the prototype system, V_{DDIO} varies at most by 0.2 V/μs, and the output tracks V_{DDIO} to within 30mV. The hold circuit adds negligible delay, and increases the energy consumption within each pad by only 1%.

The ESD protection is comprised of 500 μm^2 diodes to ground and V_{DDIO}, which is the recommended size according to the process manual [hp95]. The diodes provide the primary and only ESD protection, and were validated by simulating the human-body model (HBM) for discharge, which generates a 2kV charge pulse [hp95]. The current pulse peaks at 1.33A, and with 5Ωs of series resistance, the voltage rises to 9.1V, which is under the failure limits. To prevent the ESD diodes from turning on, ground and power bounce must be limited to less than 0.5V. Both input nodes have a 100 Ω poly resistor for isolation to prevent gate-oxide breakdown.

The input level conversion is a simple inverter powered at V_{DD}. A latch is used to maintain logic state at the output of the pad, and the output inverter is sized to drive a 1pF load.

7. 2. 8. 2 Ground & Power Bounce

The low impedance epitaxial p+ layer of the process essentially shorts out all of the chip grounds, creating a single ground network. Bypass capacitance can minimize bounce on the power lines, but ground is the global chip reference voltage and must be stabilized. To minimize absolute bounce, the ground network should contain as many pins as possible. A total of 28 pins were allocated, or 21% of all the package's pins. Simulations show that with the maximum number of I/Os switching, the ground bounce is between -540mV and +410mV at the maximum V_{DDIO} of 4V, which provides sufficient margin to prevent the ESD diodes from turning on (0.6-0.7V).

While bypass capacitance minimizes the V_{DDIO} bounce relative to ground to minimize I/O delay variation, it is also important to minimize the V_{DDIO} bounce in absolute terms to prevent the ESD diodes from turning on. Simulations demonstrated a worst case bounce of -600mV and +450mV at 4V for ten V_{DDIO} pins, with 2nF of on-chip bypass capacitance. The primary cause for the drop is the speed at which the output drivers are turned on. The device sizes were reduced to slow down the rise/fall times by 4x, but have fast turn off times.

The processor core is much more sensitive to power bounce due to timing considerations. A global reduction in V_{DD} will not affect functionality, as all the processor circuits' delay will scale appropriately. However, if only a part of the chip experiences a reduction in V_{DD}, timing violations leading to functional failure may occur. Thus, 16 pins were allocated to V_{DD}, and evenly distributed around the chip periphery. In addition, 16nF of bypass capacitance on V_{DD} is spread throughout the chip to minimize localized V_{DD} variations.

7. 2. 8. 3 Global Routing

The RC delay on global signals is only critical on the processor bus, which has as little as 8ns to operate within at 4V. The RC delay product goes up with the square of

the wire length, and becomes significant above 3.5mm for *Metal2*, and 7mm for *Metal3*. To provide sufficient margin at the maximum voltage of 4V, the RC delay must be kept below 500ps for all signal routes. The resistance is lowest on *Metal3*, which must be used for all long routes, due to its 50% lower RC delay. The program routeCap was written to calculate the RC for varying widths given a constant pitch, and report the width and space required for the wire to meet the maximum RC delay constraints. The longest route at 12mm required twice minimum width and spacing.

To eliminate Miller capacitance from adjacent parallel lines, the input and output busses are interleaved. Since they do not transition at the same time, any wire's nearest neighbors will not be switching concurrently, thereby negating the Miller effect.

7.3 Regulator IC

The primary function of the regulator IC is to convert a desired frequency value from the operating system into an output V_{DD} value which operates the processor at this desired frequency. This section gives only a brief overview of the architectural implementation, as the regulator chip is described in detail elsewhere [stra98].

7.3.1 Architecture

The architectural block diagram of the regulator is shown in Figure 7.27. The *LoadM* and *DataM* signals from the processor chip transmit the digital desired frequency value serially, which is then reconstructed as a 7-bit word, M. The clock signal, f_{CLK}, originates from the processor chip's internal VCO, and is converted to a digital word via a counter, which is then subtracted from M to calculate the frequency error, F_{ERR}. To minimize energy consumption, the entire frequency detector is operated at the variable voltage, V_{DD}. The loop filter level-converts F_{ERR} from V_{DD} to the fixed battery voltage, V_{BAT}, which is also used to power the rest of the regulator circuits. The filter generates the power MOSFETs' timing signals (*P_on*, *N_on*), and the FET driver

block converts these timing signals into the actual power MOSFETs' gate input signals. The buck converter, consisting of the power PMOS and NMOS, as well as the external LC tank, converts V_{BAT} into V_{DD}, which is then sent back to power the processor chip. Auxiliary circuits, including the current comparators and the start-up circuits, provide the control and limiting circuitry for proper operation.

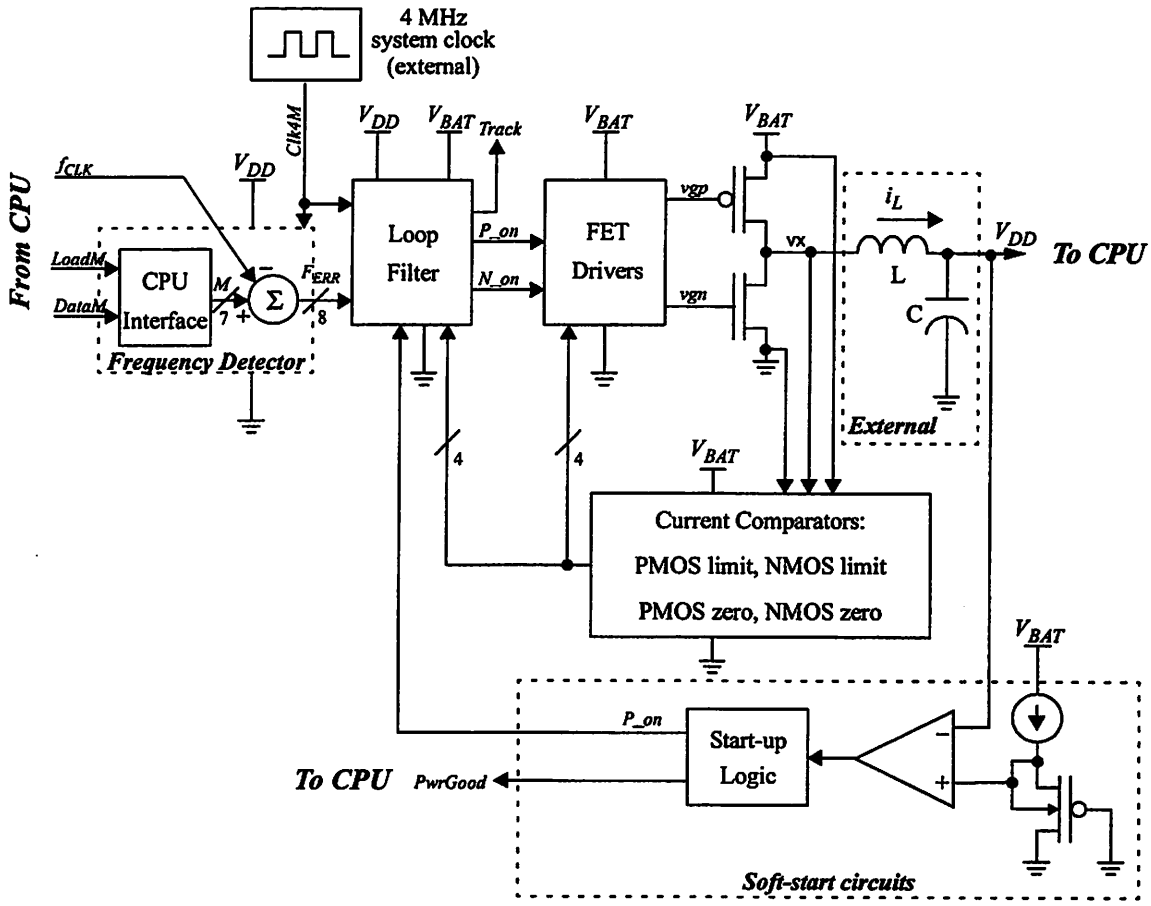


FIGURE 7.27 : Regulator Architecture [stra98].

7. 3. 1. 1 Frequency Detector

The frequency detector, shown in Figure 7.28, has a relatively simple implementation. A counter and register transforms the "analog" clock signal into a digital measure of the clock frequency in MHz. A shift register converts from serial to parallel the desired frequency sent from the processor chip, and this frequency is latched as the signal *M*. While the regulator is actively changing V_{DD}, *M* must remain

constant. Thus, during this tracking period, the *Track* signal remains high, and blocks a new value of M from being latched. Once *Track* goes low, the new value of M is loaded into the register, and the regulator goes back into tracking mode, and begins adapting to this new value of M.

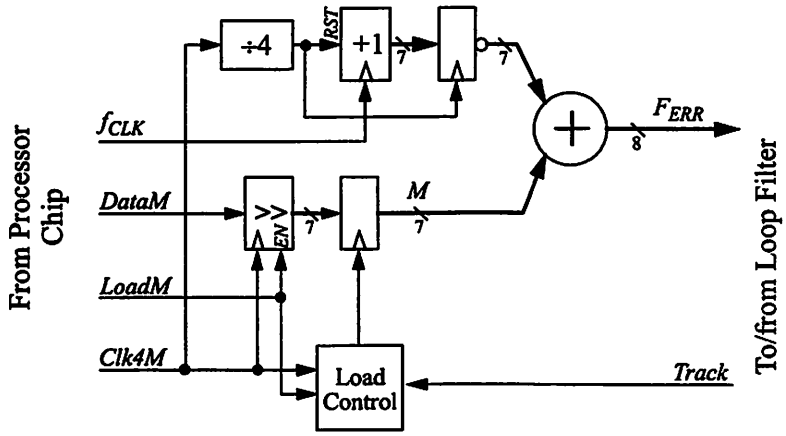


FIGURE 7.28 : Regulator Frequency Detector.

To optimize low-voltage conversion efficiency, the frequency detector circuits all operate at the voltage V_{DD}. Since the detector is the only block within the regulator that is always operating, varying its supply voltage will scale the converter's energy consumption with the desired frequency level.

7. 3. 1. 2 Loop Filter

The loop filter translates F_{ERR} into an update command for the buck converter, and implements a hybrid pulse-width pulse-frequency modulation (PWM/PFM) algorithm to provide good conversion efficiency across a broad range of output voltage and current loads. It is responsible for hand-off between regulation and tracking modes, as described in Section 3.2.

The loop filter block diagram is shown in Figure 7.29. While the converter is in tracking mode, the input register is actively latching the current F_{ERR} value. The shifter, adder, and Ton block implement the PWM part of the algorithm, by calculating the variable conduction interval for the power MOSFETs. The intermediate four-bit

digital word, *update,* is:

$$update = FF + 2^{-g} \cdot F_{mag} \qquad \text{(EQ 7.1)}$$

which contains a gain term, implemented as a binary shifter, and a feed-forward component for DC compensation. Both the gain (g) and feed-forward (FF) are a function of M, which corrects for the non-linear V_{DD} to f_{CLK} conversion in the processor's VCO. The Ton block then uses the update signal to determine how many 250µs clock cycles to keep the power MOSFETs on for (via *P_on* and *N_on*) to provide the required charge pulse given F_{ERR} and the desired frequency value.

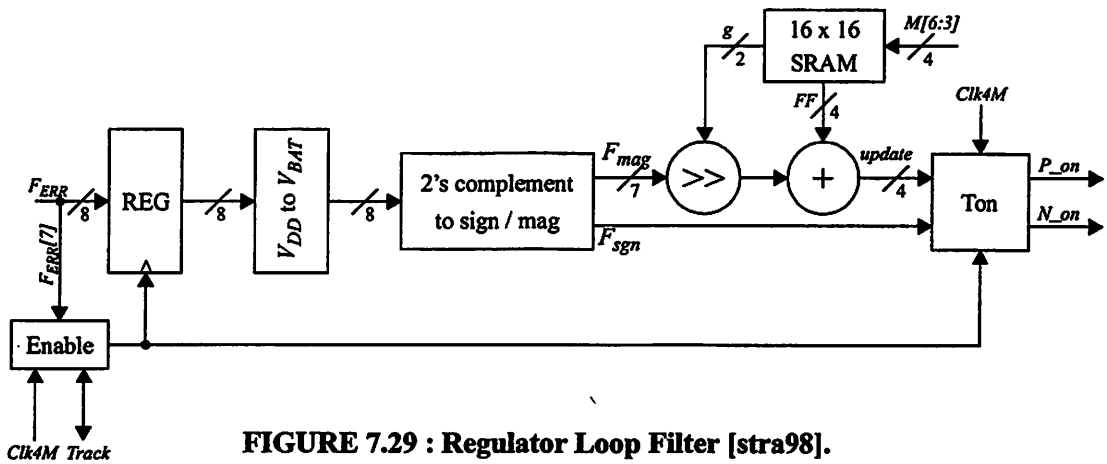


FIGURE 7.29 : Regulator Loop Filter [stra98].

In regulation mode, the loop filter's PFM aspect of the algorithm only activates the PWM circuits for positive F_{ERR}. Negative F_{ERR}, indicated by F_{sgn}, implies that V_{DD} is too high, so the loop filter suppresses the charge pulse in the current cycle and allows the current load of the microprocessor to reduce V_{DD}.

7.3.1.3 FET Drivers

The FET drivers buffer the gate enable signals of the loop filter (*P_on* and *N_on*) to drive the large gates of the power MOSFETs. To optimize conversion efficiency, the power MOSFET's are binary-weighted to provide four levels of device size based M. The FET drivers block is responsible for enabling the requisite number of power MOSFET devices.

7.3.2 Pin-out

The regulator chip was designed for a 68-pin LDCC package, whose pins are described in Table 7.10.

TABLE 7.10 Converter Chip Pad Breakdown (68-pin LDCC package).

Signal	i/o	Pads	Supply	Description	Pin Numbers(s)
VX		10		Power FET switching node	1-6, 65-68
pV_{DD}		6		Power FET V_{BAT}	10-15
pGND		6		Power FET ground	55-60
V_{BAT}		3		Digital supply at V_{BAT}	18, 48, 51
V_{DD}		2		Digital supply at V_{DD}	20, 31
GND		5		Digital ground	19, 21, 30, 47, 50
aV_{DD}		1		Analog supply at V_{BAT}	38
aGND		2		Analog ground	36, 41
vgp[a]	o	1	V_{BAT}	Power PMOS gate	54
vgn[a]	o	1	V_{BAT}	Power NMOS gate	52
Clk4M	i	1	V_{BAT}	4 MHz, 50% duty clock input	53
PORB	i	1	V_{BAT}	ResetB signal	49
PwrGood[a]	o	1	V_{BAT}	Indicates completion of soft-start	46
RAM_dOUT	o	1	V_{BAT}	Data from converter to EEPROM	16
RAM_cs	o	1	V_{BAT}	EEPROM enable	17
RAM_dIN	i	1	V_{BAT}	Data from EEPROM to converter	22
RAM_clkout	o	1	V_{BAT}	EEPROM 125 kHz clock	23
DataM	i	1	V_{DD}	Serial load of *M*	24
LoadM	i	1	V_{DD}	Enable serial load of *M*	25
Track[a]	o	1	V_{BAT}	Indicates status of control loop	26
fclk_out[a]	o	1	V_{DD}	Decoded VCO output	32
EnExtClk	i	1	V_{DD}	Enable full-swing VCO input	33
ExtClk	i	1	V_{DD}	Full-swing VCO input	34
fclk_in	i	1	V_{DD}	Low-swing VCO input	35
Vref	i	1		Low-swing reference voltage	37
Vfb	i	1		V_{DD} Kelvin sense	39
ibias	i	1		Attach 10 µA pull-down source	40
Ilim_1A	i	1	V_{BAT}	Sets 1 A or 0.5 A current limit	44
TESTenable	i	1	V_{BAT}	Sets test mode	45

a. Output is enabled only when *TESTenable* = 1.

The regulator die, shown in Figure 7.30, is 1.6mm x 3.4mm in a 0.6µm 1P3M CMOS process.

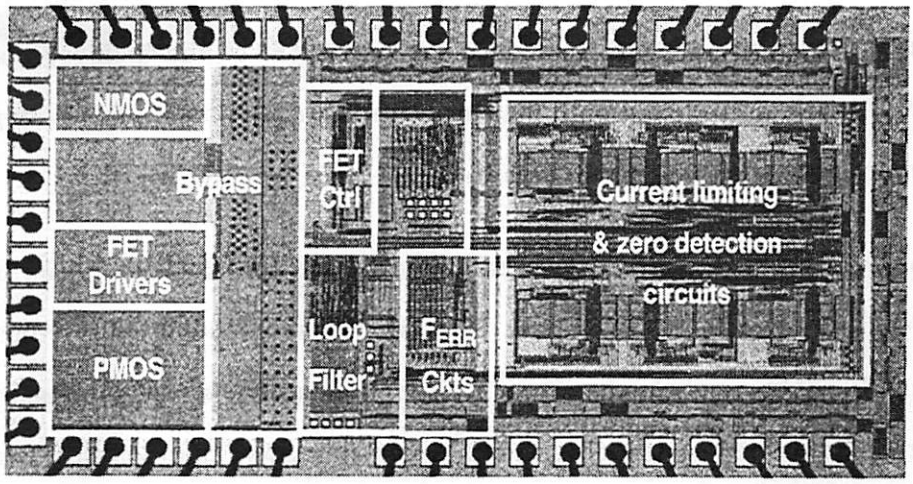


FIGURE 7.30 : Regulator Chip Die Photo.

7.4 System Bus

A key goal of the prototype processor system was to demonstrate DVS at the system level. Existing bus topologies could not be used, nor could commodity memory chips, since they require a fixed voltage. Thus, the processor system bus was designed in its entirety for energy efficiency, without having to conform to legacy standards.

The long-term vision is to utilize low-swing bus transceivers (Section 6.3) to make the energy consumed driving the bus completely negligible. To aid in debugging the prototype system, however, this option was not implemented, as it required the ability to operate the processor bus at a fixed 3.3V to use commercial test equipment. However, always operating the bus at 3.3V would completely diminish the energy savings while the processor is operating internally at low voltage. Thus, the processor bus itself was designed to be voltage scalable, with the option of running at fixed voltage, when necessary, for debugging purposes.

7.4.1 Overview

Traditional memory systems use 8-bit or 16-bit memory chips so that for a 32-bit memory access, either two or four chips need to be activated. To improve system

energy-efficiency, the memory chip's data bus is 32-bits wide, so that only one chip needs to be activated per access. To reduce the pin count of the memory chips, the address and data are multiplexed onto the same bus. This adversely affects single data bus transfers, by reducing their bandwidth by 50%.

However, the bus was designed to support burst transfers, in which multiple data words can be transferred per address. Simulation demonstrated that the bus traffic is predominantly cache-line reloads, which transfer data in bursts of eight, such that the bandwidth reduction is closer to 11%, and is acceptable given the overall reduction in system energy consumption. Thus, near peak utilization is achieved on the processor bus despite having to multiplex address and data onto the same physical bus.

7.4.2 Timing

The timing for the processor bus is shown in Figure 7.31, for both a single-word transfer as well as a burst transfer. In the first cycle that *CE* goes high, or *IOCE* in the case of the interface chip, the address is placed onto *PBus*, the *Burst* signal is asserted, and the *Write* and *Byte* signals are set accordingly. *Burst* remains high until either the second-to-last word for a read, or the last word for a write, which allows for an arbitrary-length burst of data to be transferred. *CE* remains high for the entire duration of the transaction, and is used to gate the clock within the individual chips. The *PWait* signal can be used to stall the processor bus during a transaction. The memory chips have a single wait state for a read (none is needed for a write) to allow the SRAM time to retrieve the data. The interface chip may insert a variable number of wait states depending upon how long it takes to fetch data from the external I/O peripherals.

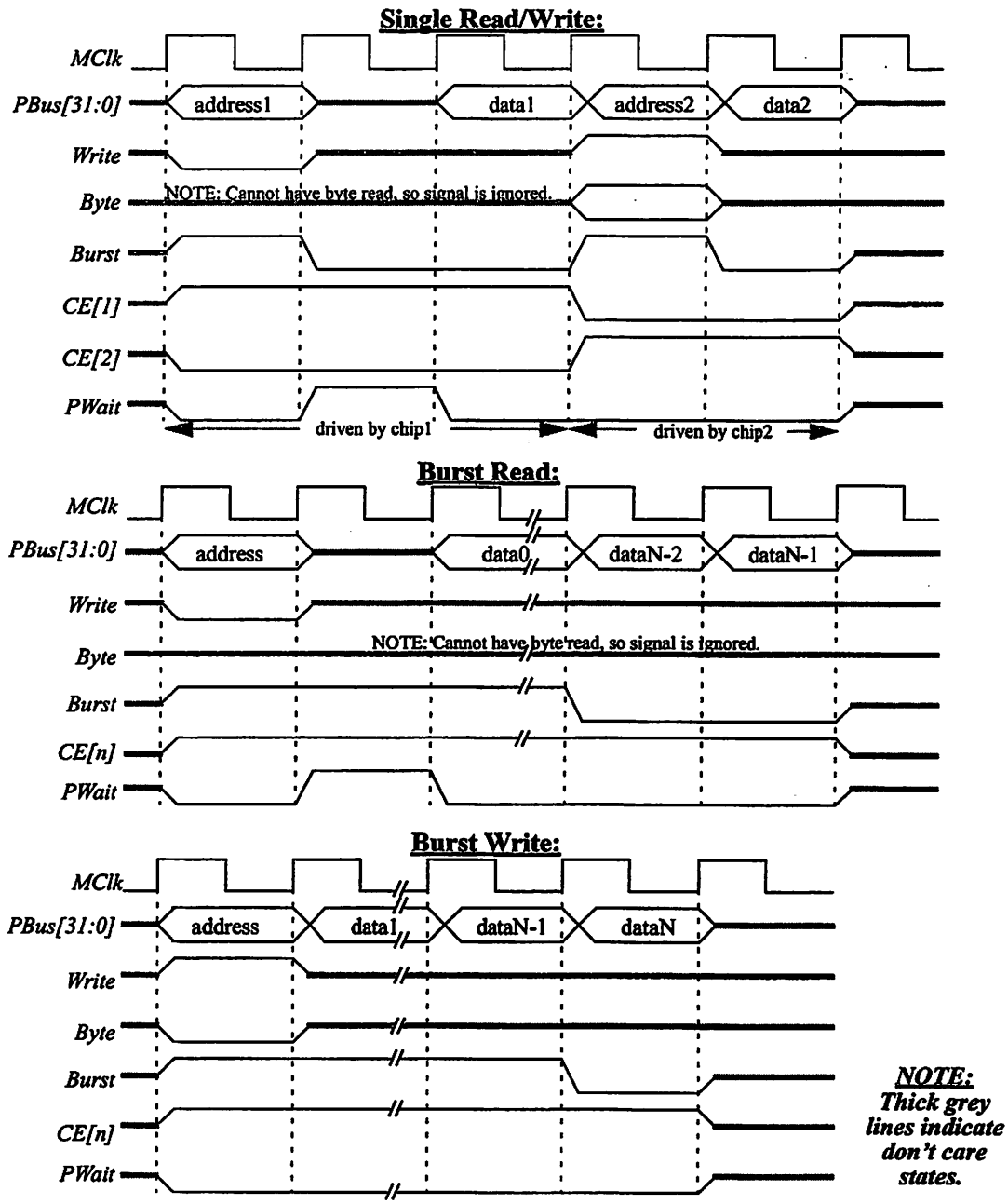


FIGURE 7.31 : Timing Diagrams for Processor Bus.

7.5 Memory IC

The memory chip is based upon the SRAM design used within the processor's cache, and was designed to be DVS compatible while optimizing energy-efficiency. The new, key design challenge was the organization of the block-level architecture and

global routing in order to minimize energy consumption. In addition, the memory chip supports split internal/external voltage sources, so that the I/O can be operated at a fixed voltage while the internal voltage varies in order to facilitate system debugging.

The SRAM die, shown in Figure 7.32, measures 9.6 x 10.4mm and contains 3.4M transistors.

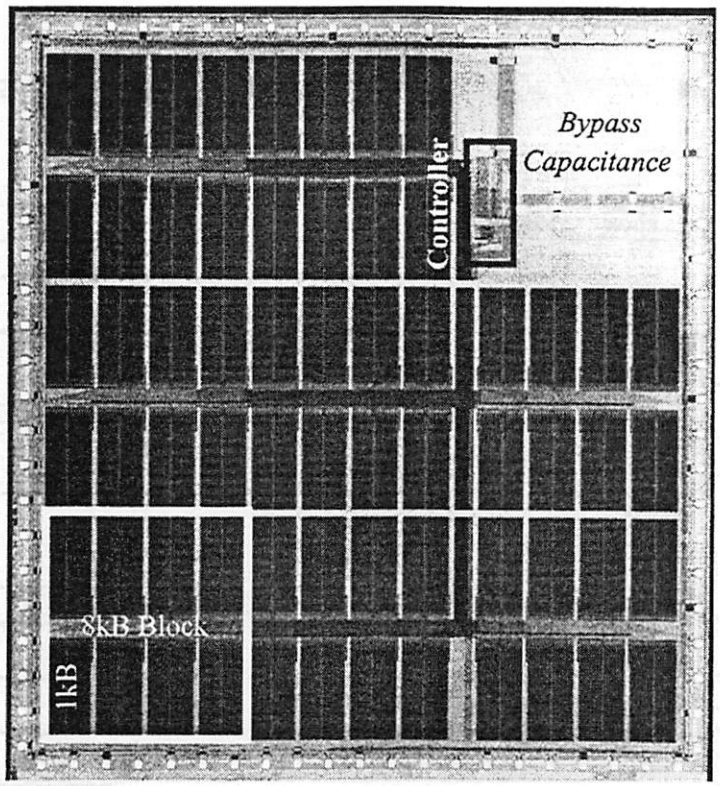


FIGURE 7.32 : SRAM Chip Die Photo.

7.5.1 Architecture

The total SRAM chip size of 64kB was set strictly by die size limitations. The basic SRAM block size was set to 1kB to provide a balanced trade-off of area efficiency (78% utilization) and energy consumption (50 pF/access). A flat hierarchy would be prohibitively expensive due to the large amount of capacitance on the bitlines, and the enormous drivers required within each SRAM block to drive this bus. Thus, a two-level hierarchy was chosen, in which the blocks are organized into an 8kB module, which was

then replicated eight times for a total of 64kB, as shown in Figure 7.33.

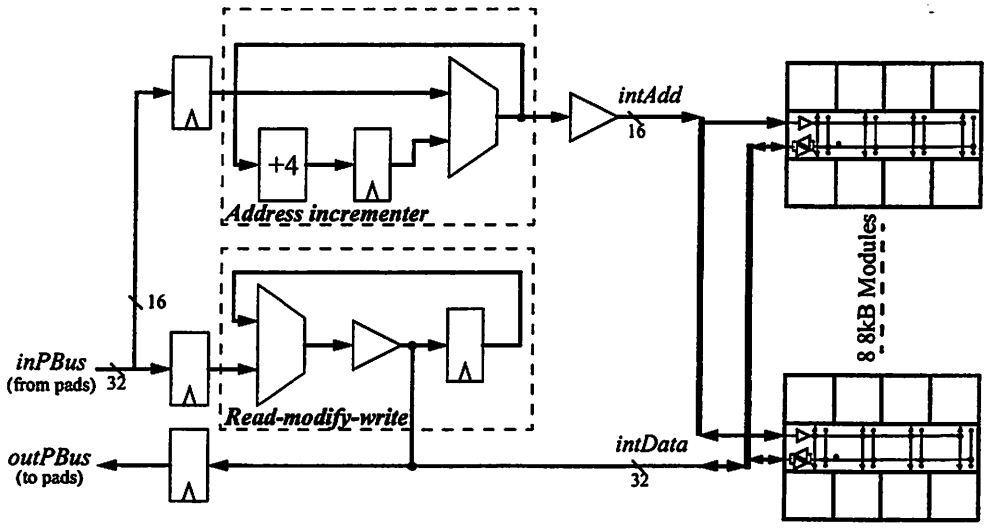


FIGURE 7.33 : SRAM Architecture.

The controller is responsible for interfacing with the processor bus, and contains additional circuitry to increment the internal address for burst-mode accesses, and to allow read-modify-write operations for byte writes. Also, the controller provides the SRAM control signals which get routed to each module. Since the address and data arrive on the same bus, only the lower 16 bits which contain the address information are routed to the address incrementer block.

7. 5. 1. 1 Operation

The internal timing of the SRAM chip is shown in Figure 7.34. The address is not available on the internal address bus, *intAdd*, until almost the end of the first cycle, which necessitates the SRAM chip to always assert the *PWait* signal for one cycle until the first data word has been read. Subsequent reads can be performed without the need for asserting *PWait*, such that an eight-word cache line requires only ten cycles to transfer across the processor bus. There is no need to stall the processor bus during either a word write, or a byte write, as shown in the lower two timing diagrams.

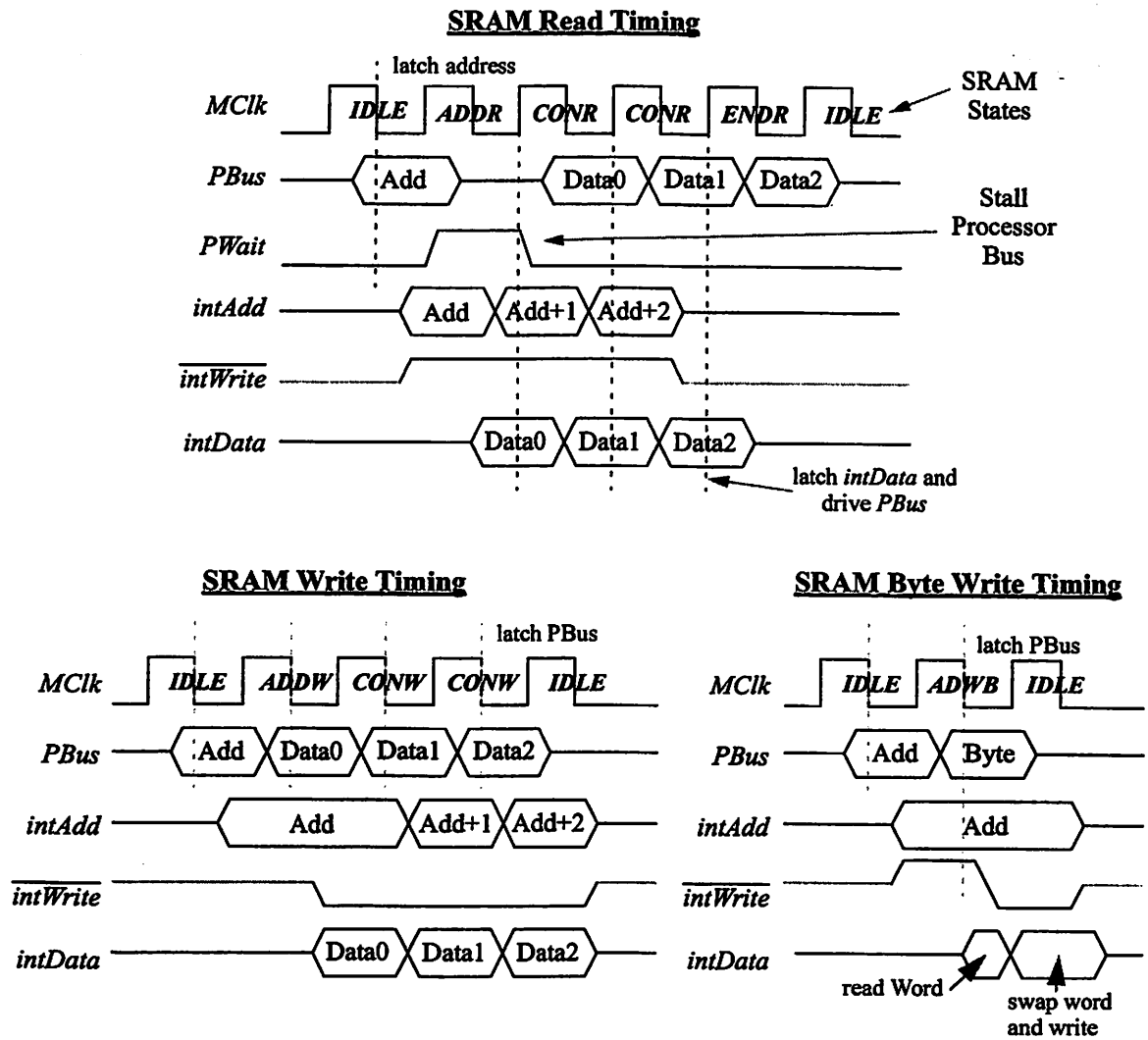


FIGURE 7.34 : Internal SRAM Timing.

7. 5. 1. 2 SRAM Module

The basic 1kB SRAM block in the module is essentially the same which was used in the processor cache and is replicated in the SRAM chip, with the addition of an address decoder. Since the 8kB module size is the same as the cache partition of 8kB, the output drivers see the same load and did not require redesign. The capacitance/cycle of the 1kB SRAM block is 50 pF/cycle for both read and write operations, with another 10 pF/cycle required to drive the interconnect within the module.

To reduce loading on the global busses and control signals, they are buffered before driving the local module interconnect. Since the data bus is bidirectional, it

contains bidirectional transceivers which switch direction depending on whether it is a read or write. A potential hazard arises for read-modify-write operations required for byte writes, which switch the direction of the transceivers between cycles. To eliminate unnecessary short-circuit current, the enable signals have fast de-assertion times, and slow assertion times to ensure that either the local module data bus, or the global bus, *intData*, are not driven by two different transceivers at the same time. Furthermore, to reduce unnecessary switching activity, the direction of the transceivers are left in whatever the last state was, so that there is no default direction that they always switch back to.

7.5.2 Energy Consumption

Performing a single read or write has an effective switched capacitance of 150-200pF over three and two bus clock cycles, respectively. Additional words in a burst read or write contribute approximately 75pF per word, and are much less because the internal address and control lines remain driven from the first data access. A byte write operation has an effective switched capacitance of 250pF, due to the combination of an SRAM read and write to complete it. The most common type of operation is a cache-line reload, which requires 725pF over ten bus cycles. Thus, if the SRAM chip is constantly active, it contributes a maximum of 36 pF/processor-cycle (there are at least two processor cycles per bus cycle), which is only 11% of the 320 pF/cycle consumed by the processor chip while it is active. In practice, the average capacitance/cycle will be lower since the SRAM is not constantly active. Thus, the SRAM was successfully designed to have minimal impact on total system energy consumption.

7.5.3 Package

The SRAM die was placed into an 84-pin QFP package. There are 39 signal pins, with 38 pins required for the processor system bus and one pin required the reset signal. A total of 45 supply pins ensure that the ground and supply bounce is maintained

well below a diode-drop of 600mV. In addition, there is 4.3nF of bypass capacitance on the I/O power supply, V_{DDIO}, and 9.6nF of bypass capacitance on the core power supply, V_{DD}. A 68-pin package would have been sufficient in providing enough supply pins to keep the ground and supply bounce within tolerable levels, but a larger pin-out package was needed in order to have a sufficiently-sized cavity given the large die size.

TABLE 7.11 SRAM Chip Pad Breakdown (84-pin QFP package).

Signal	i/o	Pads	Supply	Description	Pin Number	I_{MAX}
gnd!		28		Single ground	5, 11, 13, 20, 26, 32, 34, 41, 43-45, 47-52, 54, 55, 58, 59, 61, 62, 64, 68, 74, 76, 83	
V_{DD}		8		Internal voltage (1.2-3.8V)	3, 18, 39, 46, 53, 60, 66, 81	
V_{DDIO}		8		I/O voltage (1.2-3.8V)	1, 8, 16, 23, 29, 37, 71, 79	
V_{BAT}		1		ESD voltage for *nRst*	56	
MClk	i	1	V_{DDIO}	LPARM Processor Bus	42	
PBus	i/o	32	V_{DDIO}		2, 4, 6, 7, 9, 10, 12, 14, 15, 17, 19, 21, 22, 24, 25, 27, 28, 30, 31, 63, 65, 67, 69, 70, 72, 73, 75, 77, 78, 60, 82, 84	33 mA
Write	i	1	V_{DDIO}		40	
Byte	i	1	V_{DDIO}		38	
Burst	i	1	V_{DDIO}		36	
PWait	ot	1	V_{DDIO}		35	33 mA
CE	i	1	V_{DDIO}		33	
nRst	i	1	V_{BAT}	External hard reset	57	

7.6 Interface IC

The primary function of this chip is to connect commercial, fixed-voltage peripheral chips to the variable-voltage system bus of the embedded DVS processor system. These chips may include ROM and DRAM, as well as chips providing system I/O, such as a serial communication controller (SCC), codecs, LCD controllers, etc. A StrongArm microprocessor and a Xilinx FPGA were used to model the I/O subsystem in the prototype system, and are described in more detail in Section 7.8. To simplify the

design of interface chip, the bulk of the control FSMs to communicate with the StrongArm were pushed into the Xilinx connecting the interface chip to the StrongArm. Thus, the primary function of the interface chip is to level convert the system bus to a fixed 3.3V bus, and perform simple flow control. The level conversion occurs in the pads so that all the internal chip circuitry operates with the variable supply voltage, V_{DD}.

In a practical system implementation, this chip would be more complex in order to enable it to connect directly to peripheral I/O chips. With the controller circuitry integrated on-chip, the controller itself could be DVS compatible providing variable performance and energy consumption. Further enhancements would include having two regulator loops -- a processor core voltage/frequency, and an external memory system voltage/frequency. This would enable high-speed DMA transfers, when necessary, when the processor core is in a low-performance mode of operations.

To aid in system debugging, the processor system bus signals are always replicated on the 3.3V Xilinx-side bus. This allowed test equipment to monitor activity between the processor and main memory on the processor system bus, at a fixed voltage. In a practical system implementation, this feature would be optionally disabled in order to eliminate unnecessary energy consumption driving these signal pins when the I/O interface is not actively being used for either an I/O read/write or a DMA request.

The interface chip die, shown in Figure 7.35, measures 4.4 x 4.4mm, and contains 40k transistors, of which 5k are used by the controller implementation located in the center of the die. The chip is pad limited with its 132 I/O signals resulting in the large die size. The entire core outside the controller contains bypass capacitance used to bypass the two input voltage supplies (V_{DD} and $V_{3.3}$).

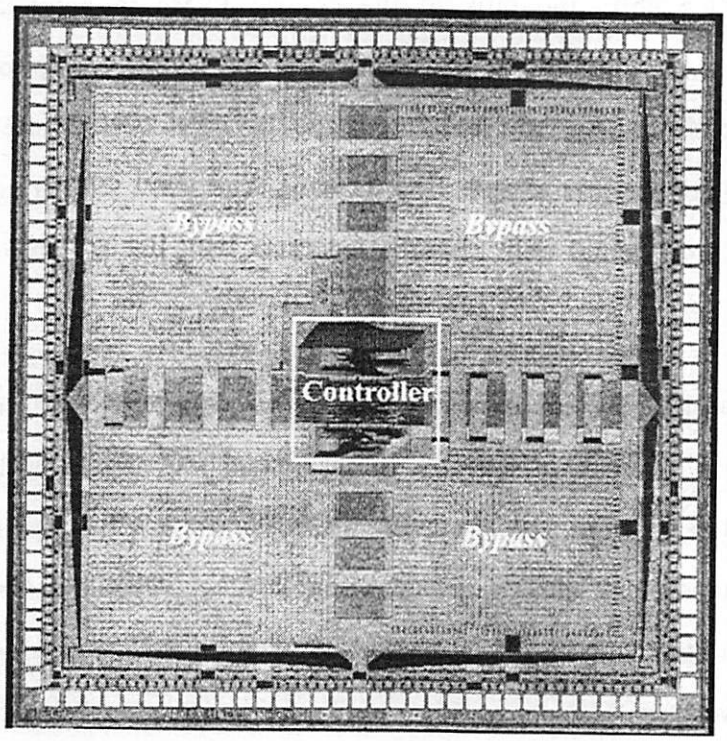


FIGURE 7.35 : Interface Chip Die Photo.

7.6.1 Architecture

The basic chip architecture is shown in Figure 7.36. When there is no active I/O or DMA request, the interface chip is in snoop mode. The bus clock (*MClk*), the processor bus (*PBus*), and the bus control signals (*Write*, *Byte*, *Burst*, *PWait*) all drive their equivalent Xilinx-side signals. All the on-chip signal paths are delay matched to maintain a constant delay shift across the Xilinx bus. To eliminate 15 unnecessary pins, the 16 memory chip enables (*CE[15:0]*) are OR-ed into a single signal, *XCE*.

For an I/O request, the processor asserts *IOCE*, which gets level-converted to *XIOCE*, and signals the Xilinx that an I/O request needs to be serviced. Because all the circuit paths forwarding signals from the processor bus to the Xilinx bus are delay matched, the Xilinx can interface to this bus in a synchronous manner since there is approximately zero relative delay shift on the Xilinx bus. This removed the need for an otherwise more costly asynchronous interface between the interface chip and Xilinx.

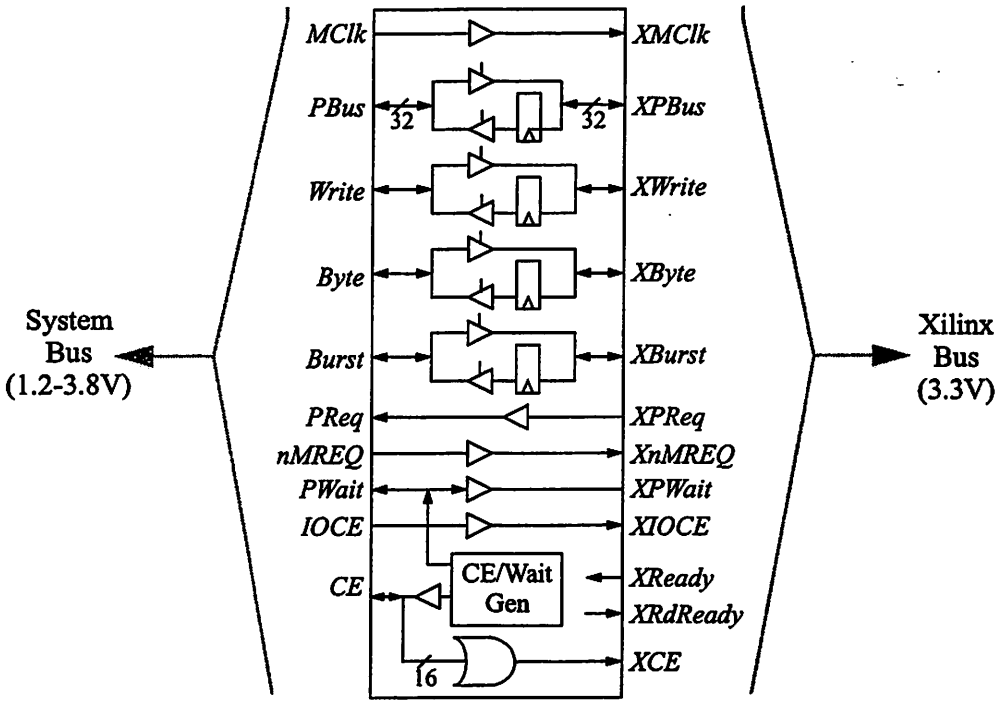


FIGURE 7.36 : Interface Chip Block Diagram.

The timing for an I/O write is shown in Figure 7.37, which demonstrates how the interface chip interacts with the Xilinx chip. The transactions get replicated from

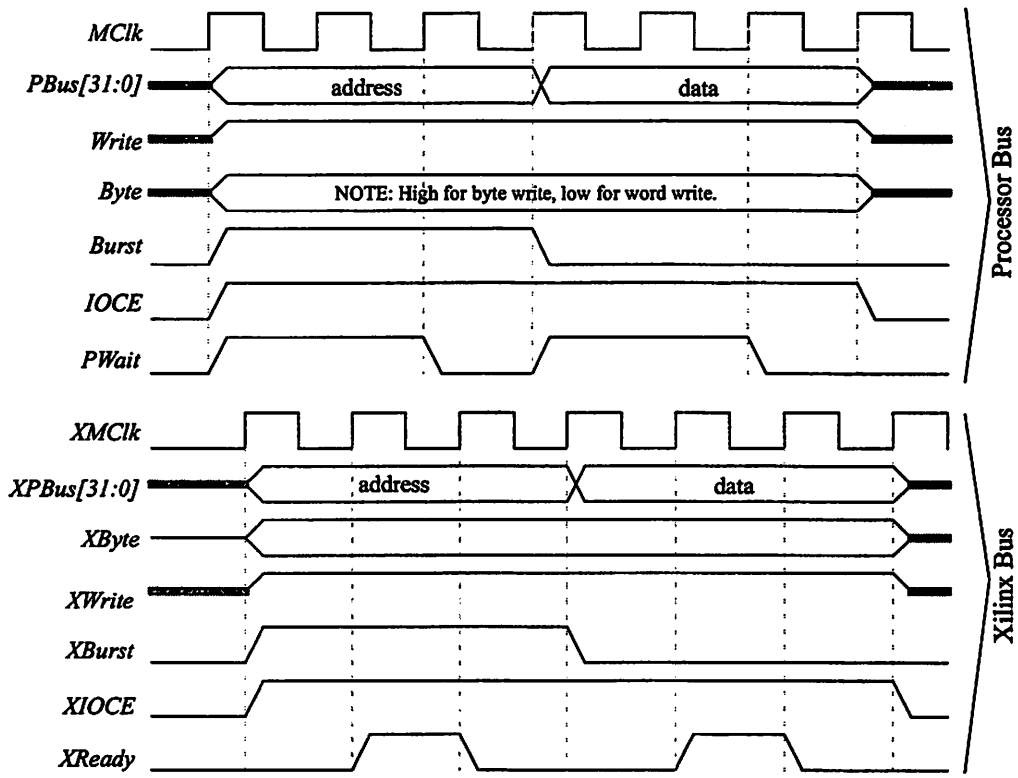


FIGURE 7.37 : Timing Diagram for a Single I/O Write Request to the Interface Chip.

the processor bus to the Xilinx bus, with the *XReady* signal providing flow control from the Xilinx back to the interface chip. By default, an I/O request will initially drive *PWait* high, stalling the processor system. Once the Xilinx has latched the address off of *XPBus*, it asserts the *XReady* signal one cycle, which in turn drives *PWait* low for one cycle, and advancing the state of the processor system one cycle. Once the data word has been transferred to the Xilinx chip, the transaction is complete. Because it can take many cycles to complete an I/O request, *PWait* is generally high for a majority of the duration of an I/O request.

On an I/O read (Figure 7.38), the *XPBus* switches direction, as indicated when the *enXPBus* signal goes low, in order to receive the desired data. Since *XPBus* gets driven by the Xilinx delay-shifted with respect to *MClk*, it is latched and driven onto *PBus* the subsequent *MClk* rising edge, requiring the 32-bit latch to hold state for one cycle. To prevent both the interface chip and the Xilinx from driving *XPBus* at the same time, the *enXPBus* signal goes low a cycle early and stays low an extra cycle. As long as the delay through the interface chip is less than the cycle time of *MClk*, there will be

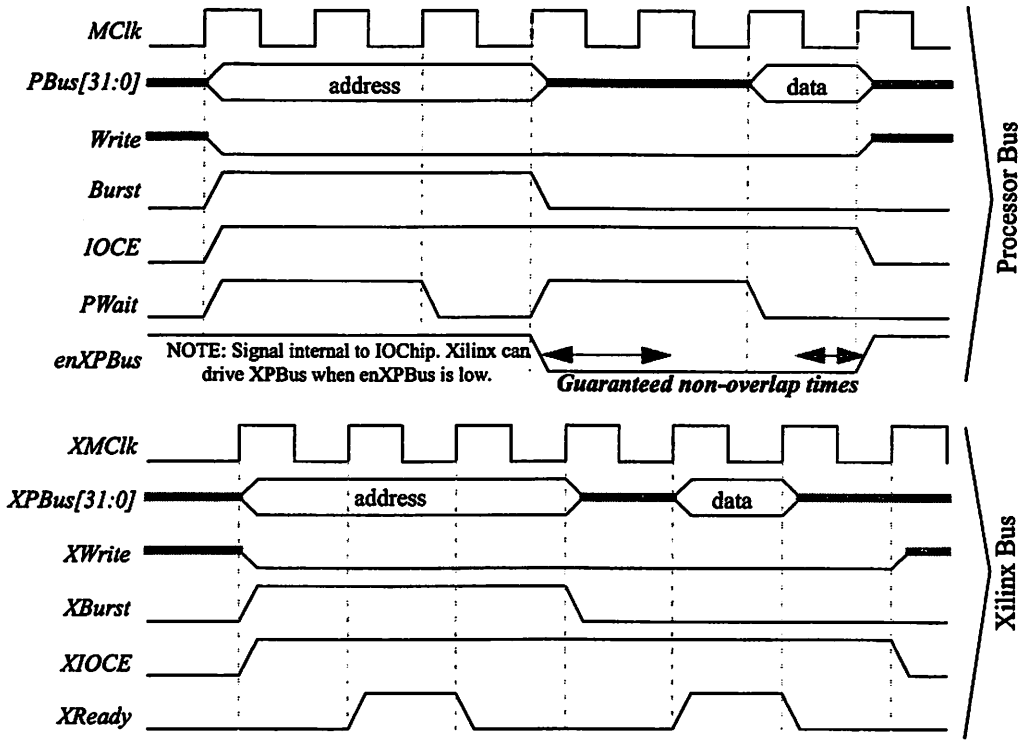


FIGURE 7.38 : Timing Diagram for a Single I/O Read Request to the Interface Chip.

guaranteed non-overlap times to eliminate this potential conflict on *XPBus*.

When the I/O subsystem wants to initiate a DMA request, the Xilinx asserts *XPReq*, which in turn asserts *PReq* and informs the processor of a pending DMA request. Once the processor has completed any outstanding bus access, it releases the processor bus and synchronously deasserts *nMREQ*, which then deasserts *XnMREQ* giving the I/O subsystem control of the bus. At the same time, the direction of the control signals is changed, and they are driven by the Xilinx via *XWrite*, *XByte*, and *XBurst*. Similar to *XPBus*, these are latched in order to resynchronize these signals with the edge of *MClk*. The Xilinx does not need to drive the *CE* signals, as they are internally generated by the interface chip, which can infer these signals by decoding the address placed on *XPBus*. In DMA mode, the *XRdReady* signal is used to indicate when the SRAM has returned the value of a DMA read request.

7.6.2 Pin Out

The interface chip was placed into a 132-pin QFP package. There are 103 signal pins, with 56 pins required for the processor system bus and 44 pins required for the Xilinx bus. An additional two pins are utilized for debugging, and one pin for the reset signal. The remaining 29 pins are used for ground and supply lines. The active circuit area on the interface chip was only approximately 2 mm^2. The large die size was necessary given the large number of pins required to interface between the two busses. The complete chip pad breakdown is given in Table 7.12.

TABLE 7.12 Interface Chip Pin Out (132-pin QFP package: 103 signal, 29 power pins).

Signal	i/o	Pads	Supply	Description	Pin Number	I_{MAX}
gnd!		16		Single ground	1,9,17,26,34,42,50,59,67, 75,83,92,100,108, 116,125	
$V_{3.3}$		6		Core/Xilinx voltage (3.3V)	7, 18, 29, 105, 117,127	
V_{DDIO}		6		I/O voltage (1.2-3.8V)	40, 51, 61, 73, 84, 94	
V_{BAT}		1		ESD voltage	102	
MClk	i	1	V_{DDIO}	LPARM Processor Bus	99	
PBus	i/o	32	V_{DDIO}		98,97,96,95,93,91,90,89,88, 87,86,85,82,81,80,79,78,77, 76,74,72,71,70,69,68,66,65, 64, 63,62,60,58	33 m
Write	i/o	1	V_{DDIO}		57	33 m
Byte	i/o	1	V_{DDIO}		56	33 m
Burst	i/o	1	V_{DDIO}		55	33 m
PReq	o	1	V_{DDIO}		54	33 m
nMREQ	i	1	V_{DDIO}		53	
PWait	i/o	1	V_{DDIO}		52	33 m
CE	i/o	16	V_{DDIO}		30,31,32,33,35,36,37,38,39, 41,43,44,45,46, 47,48	33 m
IOCE	i	1	V_{DDIO}		49	
XMClk	o	1	$V_{3.3}$	Xilinx Processor Bus	28	33 m
XPBus	i/o	32	$V_{3.3}$		27,25,24,23,22,21,20,19,16, 15,14,13,12,11,10,8,6,5,4,3, 2,132,131,130,129,128,126, 124, 123,122,121,120	33 m
XWrite	i/o	1	$V_{3.3}$		119	33 m
XByte	i/o	1	$V_{3.3}$		118	33 m
XBurst	i/o	1	$V_{3.3}$		115	33 m
XPReq	i	1	$V_{3.3}$		114	
XnMREQ	o	1	$V_{3.3}$		113	33 m
XPWait	o	1	$V_{3.3}$		112	33 m
XCE	o	1	$V_{3.3}$		110	33 m
XIOCE	o	1	$V_{3.3}$		111	33 m
Ready	i	1	$V_{3.3}$		109	
RdReady	o	1	$V_{3.3}$		107	33 m
Done	i	1	$V_{3.3}$		106	
nRst	i	1	V_{BAT}	External hard reset	101	
SwCE	i	2	V_{BAT}	Debugging pins	103, 104	

7.7 Prototype Board

The prototype system was constructed on an 8-layer 6" x 8" PCB board, with four supply layers, and four routing layers. Due to the integration of the memory and interrupt controllers onto the processor chip, few external components were required to construct the system. Extra complexity was added for features which supplemented system debugging, such as bypassing the converter chip with a fixed external voltage, and split core (V_{DD}) and I/O supplies (V_{DDIO}). The prototype system communicates to a StrongArm-based system board (Section 7.8), which emulates I/O activity, via a DB2x25 connector.

7.7.1 Architecture

The 4 unique custom ICs of the prototype system are connected as shown in Figure 7.39. The 37-bit system bus connects the processor chip to the SRAM chips and the interface chip. The nine chip enables (*CE[8:0]*, *IOCE*) are output by the integrated memory controller. An additional eight chip enables are available for SRAM chips, but were left unused in the prototype system. The interface chip communicates with the StrongArm board's Xilinx chip via a 43-bit bus, which replicates the system bus functionality with a few additional control signals. The system reset switch allows the processor system to be reset while leaving the converter actively operating, and can be used if the processor performs an illegal operation. The processor also uses 1 MHz oscillator to provide the reference frequency used by the internal real-time counter.

The converter requires additional external components. An EPROM programs up the loop filter's SRAM upon resetting the converter. Two series potentiometers provide coarse and fine-grained tuning of the chip's bias current. The 4 MHz oscillator provides the system clock to the converter, and is also used by the processor to send new clock frequency ($F_{DESIRED}$) values via *LoadM* and *DataM*. The converter also has a separate reset switch, to re-initialize its internal circuits. While the converter is being

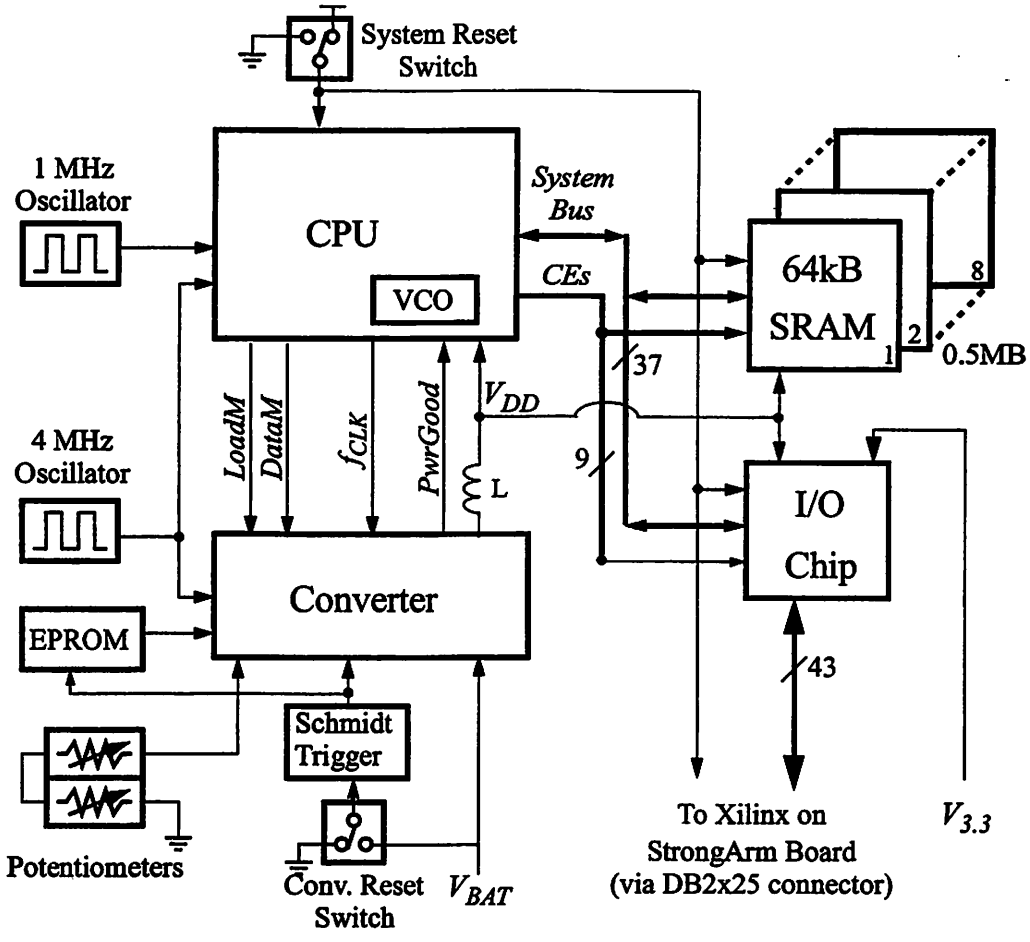


FIGURE 7.39 : Prototype Board Architecture.

reset, the *PwrGood* signal is de-asserted while V_{DD} is being re-initialized, and resets the processor chip. The converter chip also requires an external inductor and capacitor for the buck converter. The inductor was implemented via a small form-factor, SMT 4.7μH coil. The capacitor was implemented with 46 0.1μF and 0.2μF SMT capacitors placed next to the eleven chips' supply pins on the backside of the board for a combined 5.5μF of capacitance.

For an actual production system, all the external circuitry could be eliminated except for the 4 MHz reference clock frequency. The EPROM provided flexibility by allowing the loop filter's characteristics to be varied, but this was not necessary as the converter successfully operates with the initial data values. Instead, the converter's on-chip SRAM data could be hard-coded into an on-chip ROM. The potentiometers could

be eliminated via an accurate on-chip reference bias.

7.7.2 Layout

A photograph of the board is shown in Figure 7.40 demonstrating the final board layout. The board area is dominated by the eleven custom chips. Additional components previously not described are input power connectors for the supply voltages, control jumpers which provide hard-wired control settings to the processor chip, and test points which allow select internal signals from the processor chip to be monitored externally.

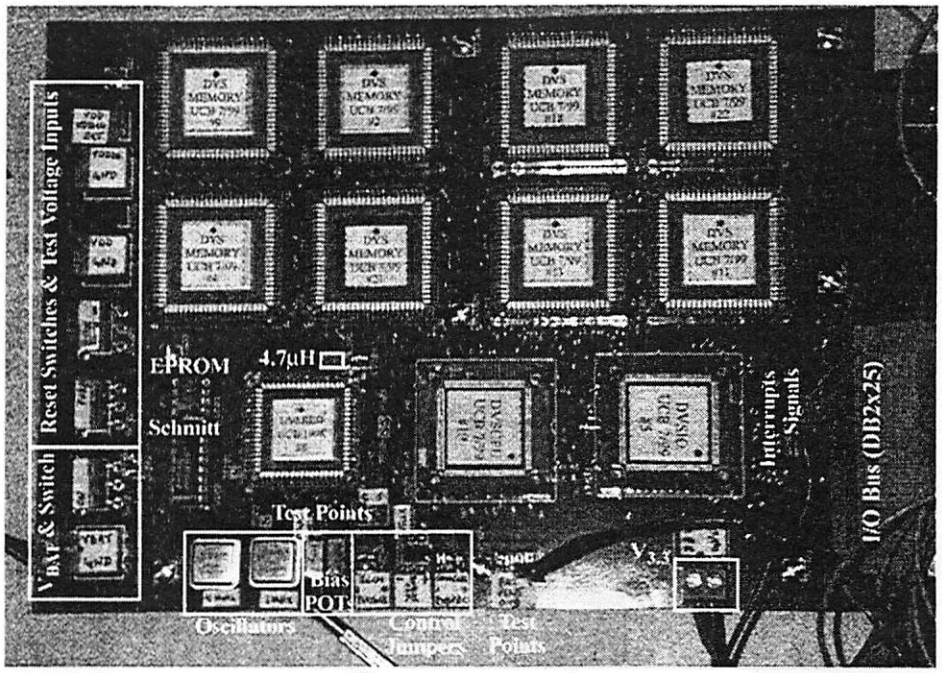


FIGURE 7.40 : Prototype Board Layout.

7.7.3 Power Distribution

A critical aspect to the design of the prototype board was managing the power distribution networks, of which there were four, as shown in Figure 7.41. The variable voltages are V_{DD}, which powers the internal circuits of the chips, and V_{DDIO}, which is used strictly for the processor bus. Jumpers at the inductor allow either the converter to drive both V_{DD} and V_{DDIO}, just V_{DD} with a fixed external voltage source for V_{DDIO}, and

neither, with external voltages supplied for both V_{DD} and V_{DDIO}. The battery voltage, V_{BAT}, powers the converter chip, and all the external components, including external control signals to the custom chips. Thus, V_{BAT} is required by all the chips to provide ESD protection. The Xilinx chip on the StrongArm board is a 3.3V part, so the internal circuits of the interface chip also operate at this voltage ($V_{3.3}$). For a future system in which the interface chip is much more complex, its internal circuits could be powered by V_{DD}, and level-converted to 3.3V at the pads to reduce system energy consumption.

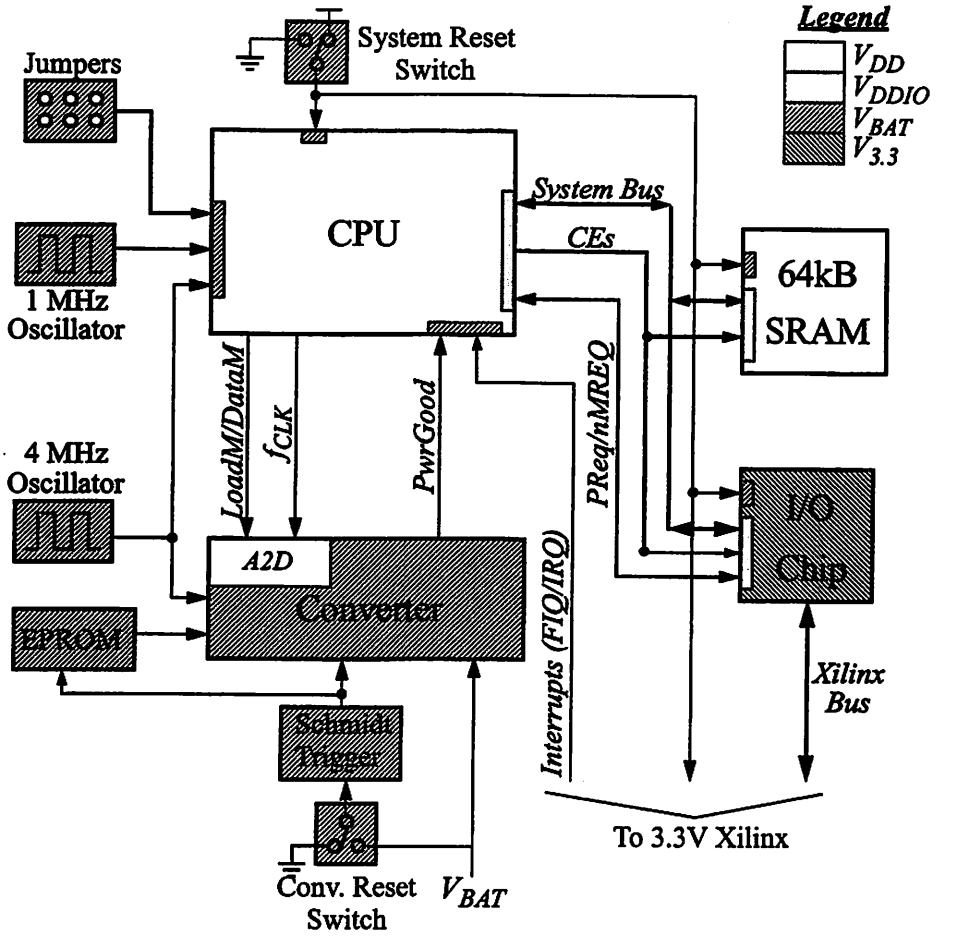


FIGURE 7.41 : Prototype Board Power Distribution.

The converter is sensitive to the parasitics on the variable voltage (V_{DD}) and the battery voltage (V_{BAT}), and were laid out so as to minimize inductor parasitics. Additional capacitance on the variable voltage power routes are tolerable if accounted for in the total capacitance placed on these nodes. The design of the power planes is

shown in Figure 7.42, in which both V_{DD} and V_{DDIO} are distributed to all the chips with minimal inductance. Likewise, those parts powered by V_{BAT} were clustered in the upper-left corner so as to provide a wide V_{BAT} power signal. The two ground planes were placed on either side of the first power plane containing V_{DD} as it is the most sensitive power line.

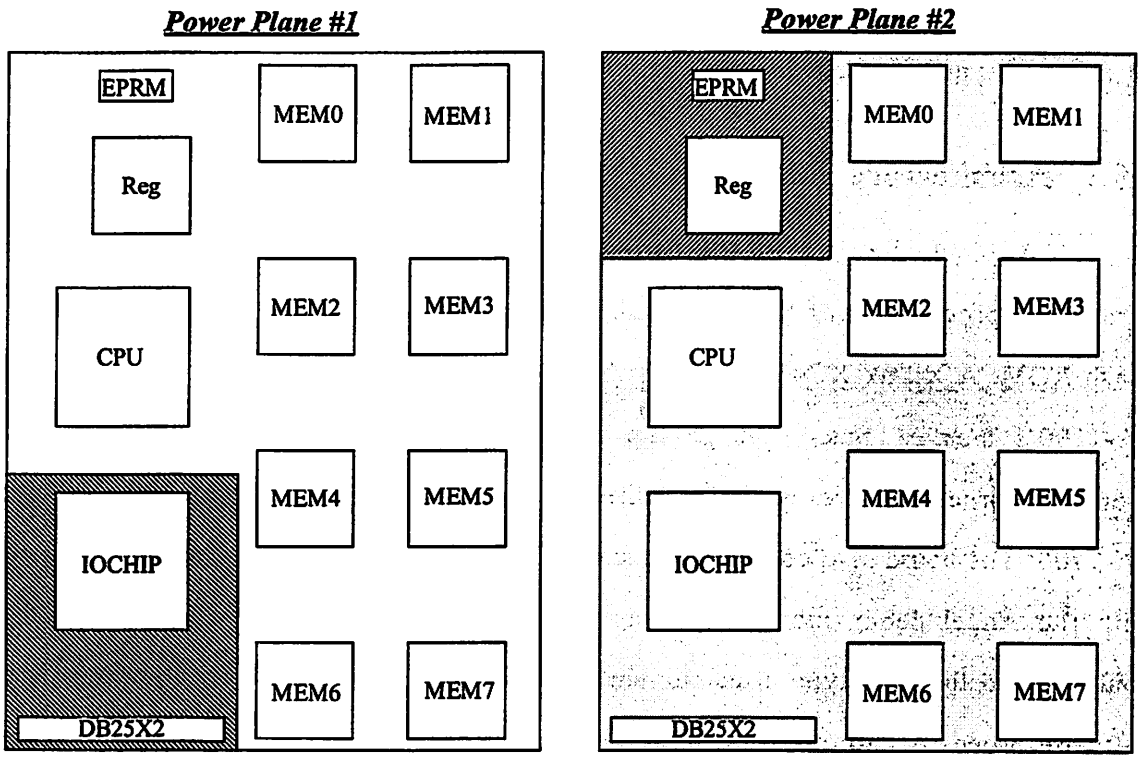


FIGURE 7.42 : Prototype Board Power Planes.

7.8 StrongArm I/O Board

The StrongArm board is a commercial development board, and used to model I/O from peripheral devices. A software approach was chosen so that I/O devices (e.g. codec, LCD, radio, etc.) could be rapidly constructed and modeled, as well as to provide low-level debugging functionality for the prototype system. The board allows all I/O output to be validated and time-stamped to ensure correct I/O output from the prototype system. In addition, the StrongArm board can generate input data at set intervals, much like any I/O device would.

The ARM programming environment provides debug and monitoring support. The current version, Angel, is used by the StrongArm board, and its predecessor, Demon, is used by the prototype processor. This debug and monitoring support consists of low-level software running on the host CPU, and remote software running on a PC or Sun workstation. The software communicates via a serial channel, which is emulated for the prototype system by the StrongArm, and allows the debugger to properly operate on the prototype processor.

7.8.1 Architecture

A block diagram of the StrongArm board is shown in Figure 7.43. The StrongArm processor is an SA-1100, which has 16MB of local DRAM and 1MB of local Flash ROM. A Xilinx XC4013 bridges the Xilinx bus from the prototype board to the SA-1100's memory bus, although the external interrupt lines (*FIQ*, *IRQ*) for the prototype processor are generated directly by programmable output pins on the SA-1100. The board also contains two serial UART ports. One is used to communicate with the Angel debug monitor running on the SA-1100, and the other is virtually connected to the prototype processor via an I/O processor emulated in software. This software processor monitors incoming data on the second UART, and routes it to the prototype system via the Xilinx, and likewise takes output data from the prototype system destined for the remote debugger, and sends it to the UART.

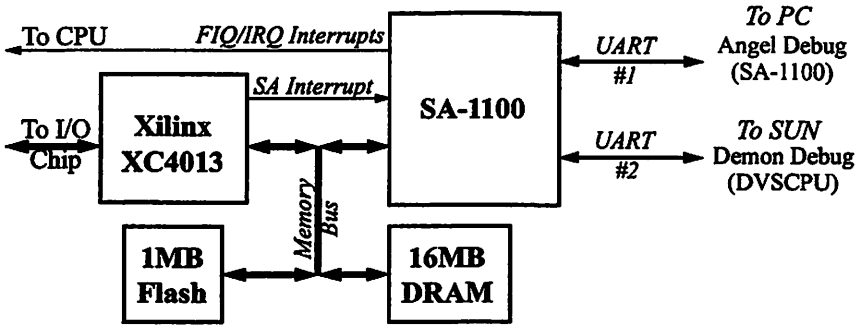


FIGURE 7.43 : StrongArm Board Architecture.

The benchmark data-sets were burned into the Flash ROM, so that the SA-1100

would not have to download the data-sets across the slow UART channel into DRAM, and would otherwise take more than 15 minutes upon a system reboot.

7.9 Software Infrastructure

To fully qualify the energy-efficiency improvement of DVS, a software environment typically found in a portable device was booted on the prototype system. This includes a real-time operating system (RTOS), the voltage scheduler required by DVS, and common application programs. The prototype system would then execute the benchmark application with and without the voltage scheduler to quantify the increase of processor system energy-efficiency due to DVS.

7.9.1 Software stack

The stack-up of the software infrastructure is shown in Figure 7.44. On the prototype processor is the low-level Demon debug monitor, on top of which sits the RTOS, the voltage scheduler and the user application programs. On the SA-1100 is the Angel debug monitor, which sits underneath the StrongArm I/O Processor (SAIOP) software program. This provides I/O support to the RTOS, and creates the virtual channel which allows Demon to communicate with the remote debugger program, armsd, running on a Sun. Another armsd program running on a PC interacts with the SA-1100 debugger.

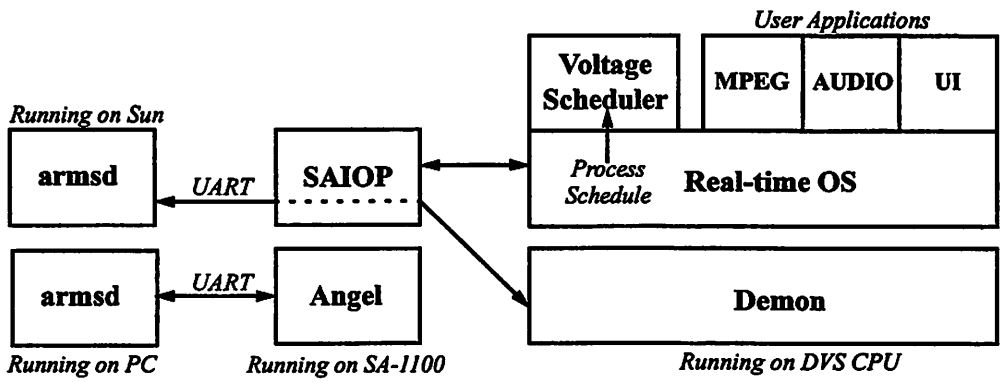


FIGURE 7.44 : Software Architecture.

The RTOS [peri98] is a custom pre-emptive multi-tasking kernel that contains a temporal scheduler and standard C library functionality. The temporal scheduler decides which task runs when using an earliest-deadline-first (EDF) policy, which is optimal for fixed speed systems [liu73]. The kernel is not cognizant of the speed setting of the processor. Whenever the temporal scheduler updates the process schedule, the voltage scheduler is executed, which is run as a separate thread on top of the kernel. The voltage scheduler analyzes the current process schedule and application deadlines to provide a voltage schedule for varying microprocessor performance. The algorithm is discussed in further detail in Section 3.5.4.

The user applications are written in C/C++ using the full C library support provided by the RTOS. The three application used in the DVS evaluation benchmark suite (MPEG, UI, AUDIO) are discussed in Section 3.6.1.

7.9.2 Software I/O processor (SAIOP)

The RTOS and user applications use address mapping, as described in Table 7.13, to specify the destination for I/O data. The SAIOP program then routes the I/O data to the desired location on the SA-1100.

TABLE 7.13 IO Space Address Mapping.

I/O Device	Address	Description
IO Channel	0x48003a00	I/O control information. Opens/closes a file or network connection, and performs flow control
	0x48002000	I/O read data.
	0x48002400	I/O read control. Provides information on channel, and µs delay until next word is to be read.
	0x48003b00	I/O write data. Data word is tagged with channel being written to.
Frame Buffer	0x58xxxxxx	Frame buffer. Writes to this space are logged in framebuffer.dat file for later verification.
	0x78xxxxxx	Frame buffer color-map. Write to this space adjust the color-map of the display device.
Debug Space	0x680001xx	Debug space. Used for low-level debugging of RTOS state.
Serial Channel	0x880000xx	Serial channel. Mimics the register set and functionality of a standard UART serial interface.

With the exception of the serial channel and frame buffer, all I/O connections are established as sockets, and time-share a single I/O channel location. Each I/O device is allocated a unique channel ID, which is used to tag all input/output data on the I/O channel for that device. Flow control is available to slow down and/or speed up the flow of data as necessary. Writes to the I/O channel are verified against the master data set stored in the Flash ROM, and reads have their data supplied by the ROM, and tagged with a delay time for which the SAIOP should wait until asserting the interrupt line to indicate that the next data word is ready.

The frame buffer is located in a separate address space, and the contents thereof are written to a file for post-execution evaluation to ensure the correct data was written to it. The debug space is used to perform low-level thread and speed tracing of the prototype processor, which aided in the debugging of the system. The SAIOP maps the virtual serial channel to the physical UART on the StrongArm board, allowing the Demon running on the prototype processor to communicate with the remote debugger on a Sun workstation.

7.10 Results

The prototype system successfully booted up on first silicon, and the entire benchmark suite was able to execute on the prototype system to demonstrate, on a real hardware implementation, the potential energy-efficiency improvement of DVS. In addition, the benchmark program Dhrystone 2.1 was run in order to measure the energy consumption in terms of MIPS/Watt, a commonly quoted measure, to compare against commercial processor implementations.

While the original design target was for V_{DD} to operate over 1.1-3.3V with a clock frequency range of 5-100 MHz, the prototype silicon failed to operate for a V_{DD} less than 1.2V. Since even the VCO failed, which consists of only CMOS pass gates and inverters, the most likely cause of failure was a much larger $|V_{Tp}|$ than specified in the

process manual (0.95V, worst case, with a 0.7-1.5V wafer acceptance range), although no test structures were on the die to verify this hypothesis. However, the prototype system successfully operated over the voltage range 1.2-3.8V, although over the somewhat lower frequency range of 5-80 MHz, as shown in Figure 7.45, demonstrating the ability of a DVS processor system to scale with widely-varying process parameters.

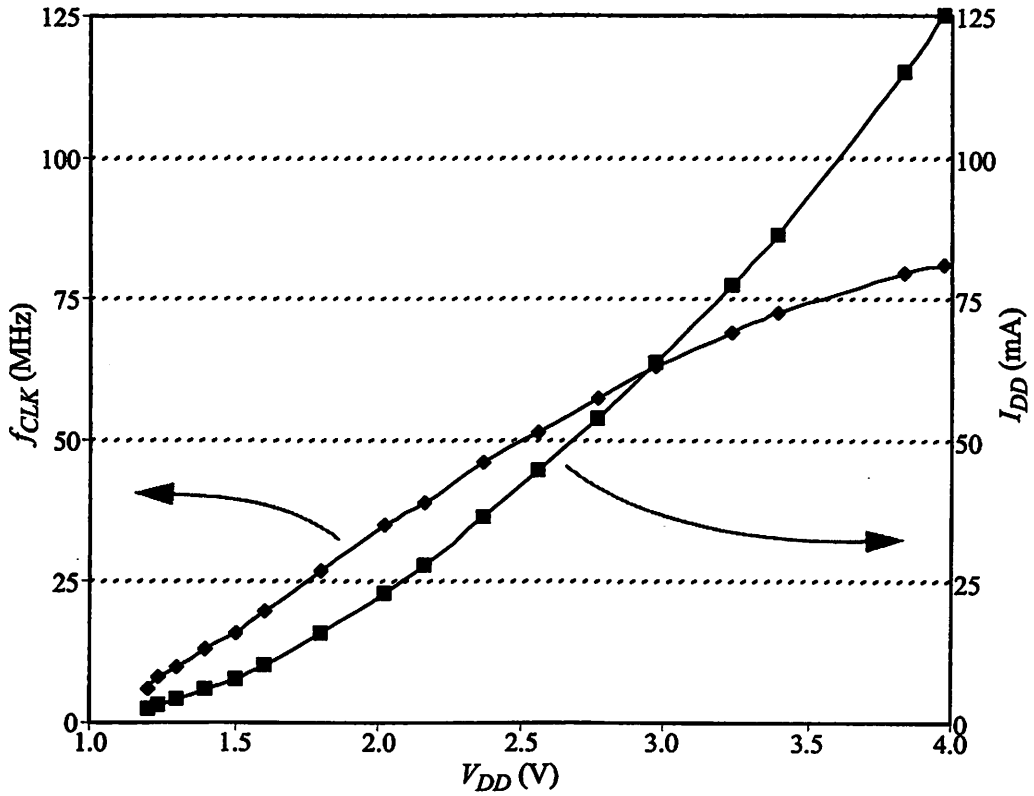


FIGURE 7.45 : Measured Clock Frequency and Supply Current vs. Supply Voltage.

7.10.1 Transient operation

Figure 7.46 shows a scope trace for the system's maximum low-to-high and high-to-low speed transitions. The V_{DD} signal transitions from 1.2V to 3.8V, then back down to 1.2V. The *Track* signal indicates whether the converter loop is in the tracking mode, in which it is actively changing V_{DD}, or in regulation mode, in which it is trying to maintain a constant V_{DD} value. This signal demonstrates that the maximum transition time is 70μs for the 5-80 MHz transition under full system load, while smaller voltage transitions can be performed in less time. During this entire transition period, the

processor system can continue to execute instructions.

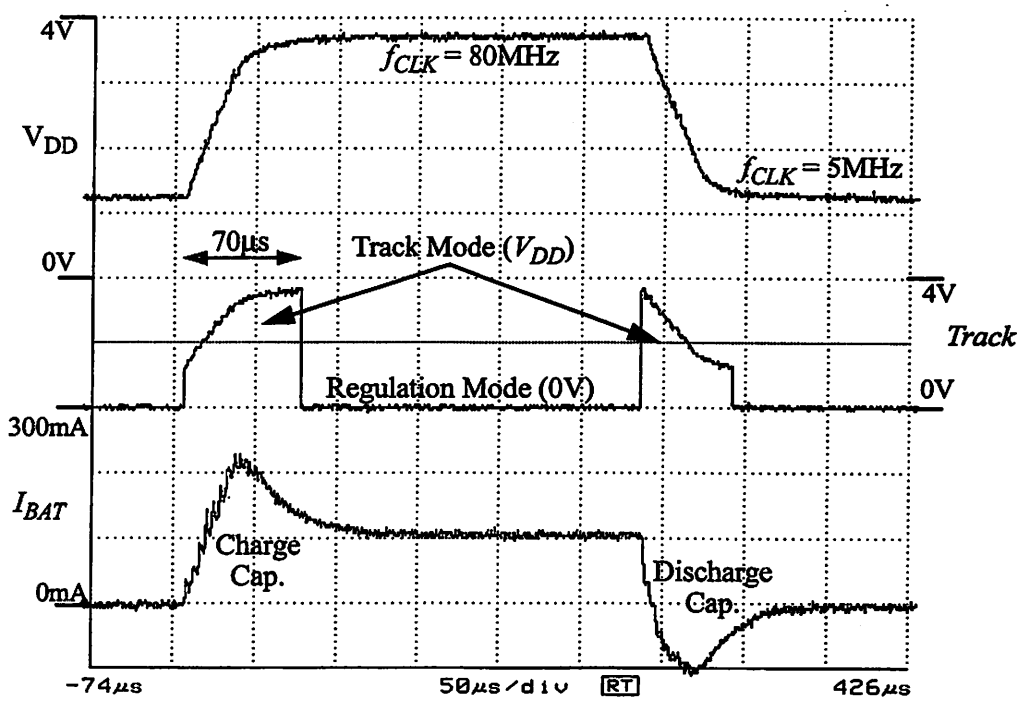


FIGURE 7.46 : Transient Response of the Converter Loop.

The decaying exponential response of V_{DD} demonstrates that the converter loop behaves much like a single dominant-pole system. In fact, V_{DD} changes to within 70% of its final value within only 25µs, because it is slew-rate limited to 0.08 V/µs.

The signal I_{BAT} is the battery current measured going into the regulator, but after the battery's bypass capacitor. There is a current spike on the low-to-high transition which is required to charge up the loop's output capacitor to the required voltage. The negative current spike on the high-to-low transition occurs because the power PMOS is removing charge from the output capacitor and placing it back onto the battery's bypass capacitor at approximately 90% conversion efficiency. The conversion loss of the loop is the transition energy, which is a maximum of 4µJ for both the low-to-high and high-to-low transitions.

To demonstrate how the converter adds charge to the output capacitor, Figure 7.47 shows a scope trace plotting the buck circuit waveforms when the converter is

regulating a constant V_{DD}. The power PMOS (M_P) is enabled first, which begins ramping up the inductor current (i_L) for a duration specified by the loop filter. At the end of the duration, M_P is turned off, and power NMOS (M_N) is turned on, which ramps down i_L until it returns to zero. When the converter is ramping up V_{DD}, the i_L pulses will be larger and more frequent, and when it is ramping down V_{DD}, i_L will be reversed in polarity and the timing of the power FETs will switch so that M_N is enabled before M_P.

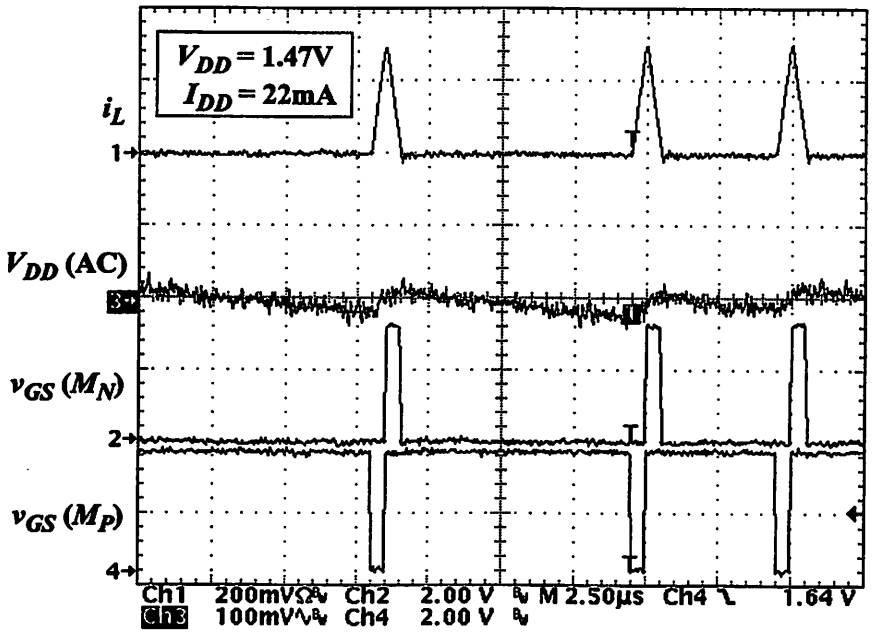


FIGURE 7.47 : Converter Waveforms in Regulation Mode [stra98].

7.10.2 Dhrystone Benchmark

The Dhrystone 2.1 benchmark is commonly used for microprocessors in embedded applications to characterize throughput in MIPS, as well as energy consumption in Watts/MIP [weic84]. This benchmark was compiled for the prototype system, so that system energy-efficiency could be directly compared against the energy-efficiency of commercial ARM implementations.

Figure 7.48 plots the prototype system's throughput versus its energy consumption for the Dhrystone 2.1 benchmark. The upper curve is for the system when

it is powered by a fixed, external voltage source, and the converter is disabled. The lower curve is for the system with the converter loop enabled. The curves are generated by running the system at constant frequency and V_{DD} to demonstrate the full operating range of the system. The throughput ranges from 6-85 Dhrystone 2.1 MIPS, and the total system energy consumption ranges from 0.54-5.6 mW/MIP. The efficiency of the converter loop, which is proportional to the gap between the two curves, ranges from 90% at high voltage to 80% at low voltage.

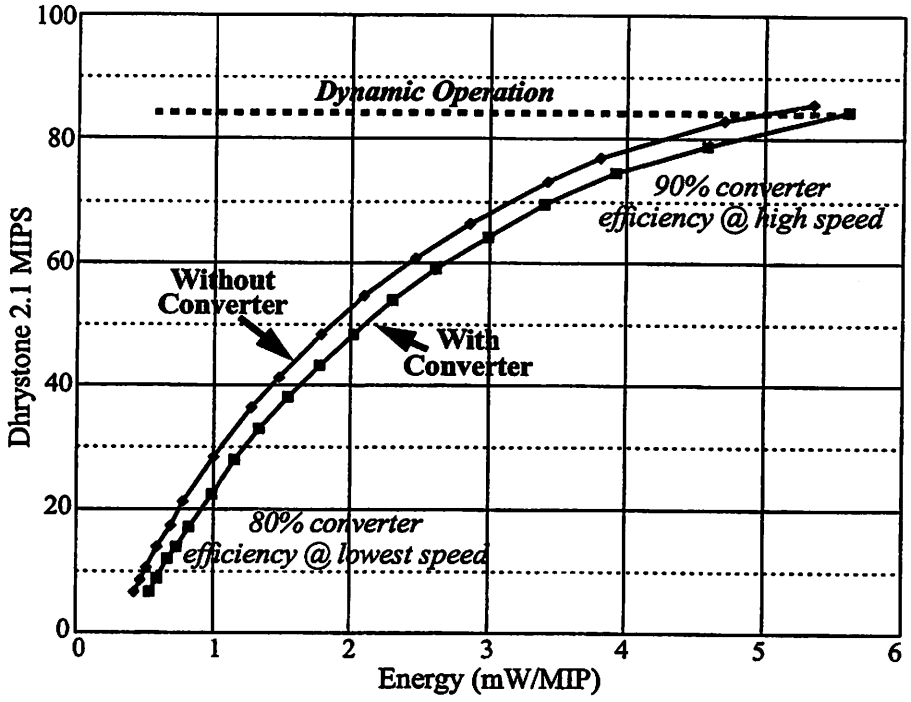


FIGURE 7.48 : Measured Throughput vs. Energy Consumption.

With DVS, peak throughput can be delivered upon demand. Thus, the true operating point for the system lies somewhere along the dotted line because 85 MIPS can always be delivered when required. When only a small fraction of the computation requires peak throughput, the processor system can deliver 85 MIPS while consuming, on average, as little as 0.54 mW/MIP.

A commonly quoted energy-efficiency metric is MIPS/Watt. The equivalent for this system would be the ratio of peak MIPS to average power dissipation because the

throughput and power dissipation can be dynamically varied. In the optimal case when peak throughput is required only a small fraction of the time, the system's average power dissipation can be as low as 3.24mW, yielding 26,200 MIPS/W. When the system is operated at constant V_{DD}, the energy-efficiency is a maximum of 1,850 MIPS/W at 1.2V.

7.10.3 Idle Energy Consumption

Because a microprocessor in portable systems idles a significant amount of time, the energy consumed while idling can become critical to the overall energy efficiency. For the prototype system, a halt instruction was implemented via a coprocessor write instruction, which asserts the *Sleep* signal. This signal effectively stops all activity by clock gating the rest of the system, with the exception of a few state registers in the interrupt controller, the external bus interface, and the real-time counters.

If the processor speed is set to 5 MHz before entering sleep, the entire system dissipates only 800μW of power, with a one cycle start-up from sleep. The latency to ramp back up to full speed upon wake-up is set by the converter loop to be 70μs, although the processor can continue operating during this ramp up period and begin immediate execution of the interrupt handler.

7.10.4 DVS benchmarks

To evaluate DVS, benchmark programs were chosen that represented software applications that are typically run on notebook computers or PDAs. Existing benchmarks (e.g. SPEC, Dhrystone MIPS, etc.) are not useful because they only measure the peak performance of the processor. New benchmarks were selected which combine computational requirements with realistic latency constraints. The three programs are MPEG, UI, and AUDIO, and are described in more depth in Section 3.6.1.

7. 10. 4. 1 Measuring Energy Consumption

To measure energy consumption of the benchmark applications, the simple circuit in Figure 7.49 was used in-line on the regulator's voltage supply, V_{BAT}. After Demon boots and the RTOS and benchmark program are downloaded into main memory, the Demon break-points the start of the application and idles at low voltage. When instructed by a "go" command from armsd, the benchmark will execute, and at the end of running, will put the processor back into idle mode at low voltage.

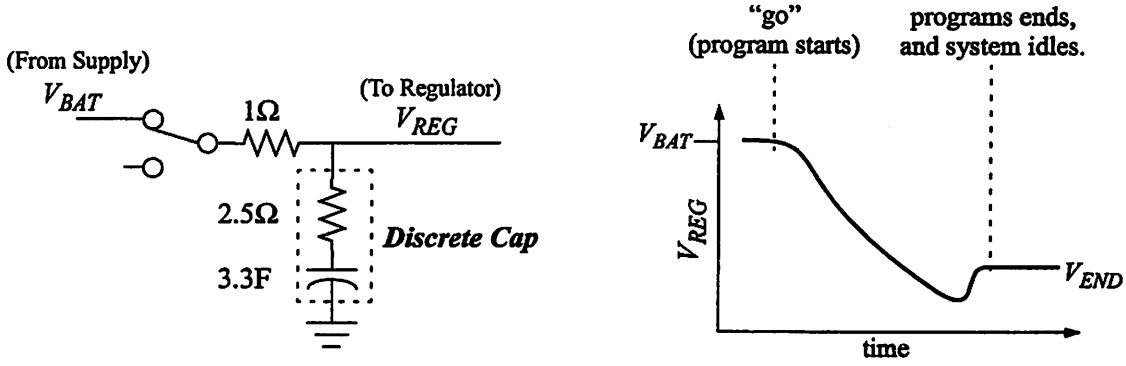


FIGURE 7.49 : Energy Measurement Circuit & Transient Response.

While Demon is booting, the switch remains closed and the capacitor maintains V_{BAT} across its terminals. At the break-point, the switch is opened, the "go" command is given, and V_{REG} roughly changes as depicted. The voltage drops due to the capacitor sourcing charge, and due to an IR drop on the intrinsic resistance of the discrete capacitor. When the microprocessor idles after completing the application, V_{REG} jumps back up a little bit due to the IR drop disappearing and settles to V_{END}. During low-voltage idle, the drop on V_{REG} is 60 μV/sec, and hence, very flat.

The energy consumption of the benchmark is:

$$\Delta E = \frac{1}{2} \cdot 3.3F \cdot (V_{BAT}^2 - V_{END}^2) \qquad \text{(EQ 7.2)}$$

There is energy loss in the 2.5Ω resistor, but at the maximum average current of 20mA, the loss is only 1.2%, which was neglected. Thus, the energy consumption of the benchmarks could be measured to within 99% accuracy.

7. 10. 4. 2 Results

Using the above approach for measuring energy consumption, the three benchmarks were first run at constant maximum throughput to measure the baseline energy consumption. They were then re-run with the voltage scheduler enabled, and had their energy consumption measured again.

TABLE 7.14 Measured Benchmark Energy Consumption (Normalized).

	Benchmark Programs		
Algorithm	MPEG	UI	AUDIO
Maximum Performance	100%	100%	100%
Optimal	67%	25%	16%
Voltage Scheduler	89%	30%	22%

Table 7.14 shows the measured system energy consumption for the three benchmarks, which is normalized to when the system is running at maximum throughput, since this is the typical operating mode of a processor system that operates from a fixed V_{DD}. The row labelled Optimal is the energy reduction when all the computational requirements are known a priori, and is an estimated value derived from simulation. The optimal values represent the maximum achievable energy reduction for these benchmarks. The last row is the measured energy consumption with the voltage scheduler enabled. As expected, the compute-intensive MPEG benchmark has only a 11% energy reduction from DVS. However, DVS demonstrates significant improvement for the less compute-intensive AUDIO and UI benchmarks, which have a 4.5x and 3.5x energy reduction, respectively. Comparing the DVS results against the optimal results demonstrates that while the voltage scheduler's heuristic algorithm has a difficult time optimizing for compute-intensive code, it performs extremely well on non-speed critical applications.

Table 7.15 shows the average power dissipation of the three benchmarks with the voltage scheduler operating. The effective MIPS/W is calculated as the ratio of peak throughput (85 MIPS) to average power dissipation, and demonstrates the achievable

increase in energy efficiency when the system is running real programs. Both the UI and AUDIO benchmarks have an average power dissipation on the order of 10mW, yielding an energy efficiency on the order of 10,000 MIPS/W.

TABLE 7.15 Measured Power Dissipation with the Voltage Scheduler.

	Benchmark Programs		
Voltage Scheduler:	MPEG	UI	AUDIO
Average Power (mW)	145	11.75	8.00
Effective MIPS/W	600	7,200	10,600

Thus, real applications, with the proper operating system support via the voltage scheduler, can achieve a significant reduction in energy consumption with DVS, thereby improving processor system energy-efficiency by up to a factor of 10x.

7.11 Comparison to Prior Art

A technique for minimizing the supply-voltage to reduce energy consumption utilizing a voltage regulator was initially proposed for digital circuits at fixed throughput [kaen90]. A replica of the critical path was used in a negative-feedback loop to set V_{DD} to the lowest possible level, while the circuits continued operating correctly given a desired clock frequency.

This technique was subsequently demonstrated on a MIPS R3900 processor core, with an integrated, on-chip, voltage regulator [kuro98]. A desired operating frequency is set externally, and the regulator outputs the minimum V_{DD} value at which the processor core can continue operating. However, the clock frequency could only be set externally, and requires a system reboot in order to change the frequency value.

This technique was enhanced to dynamically scale V_{DD} for variable-rate digital signal processing [niel94]. A variable-rate processing circuit has an input FIFO, which is monitored for how full it is. When the FIFO is near empty, V_{DD} can be reduced, and when the FIFO is near full, V_{DD} must be increased to catch-up to the input data.

Adaptive scaling was later demonstrated with an open-loop regulation approach, which used four V_{DD} values to provide faster switching transients and used dithering to approximate intermediate V_{DD} values [chan96]. More recently, an approach for dynamic voltage scaling has been demonstrated for I/O interfaces, in which V_{DD}, and the energy consumption, scales with the throughput demands on an I/O transceiver [wei00].

The work presented here extends these techniques, and combined with efforts on energy-efficient operating system design [peri00] and dynamic voltage converters [stra98], demonstrates the following:

Dynamic voltage scaling on a general-purpose microprocessor. The voltage and clock frequency of a general-purpose microprocessor, built upon an ARM8 processor core, can be dynamically varied from 1.2-3.8V and 5-80 MHz, respectively. Changes can occur dynamically, without having to halt processor operation, and occur at a fast rate (< 70μs), such that they appear instantaneous to the software executing on the microprocessor.

Direct operating-system control of supply voltage and clock frequency. A control register has been added to the processor's ISA. When the operating system writes to this register, the voltage converter will immediately change the processor's voltage such that it operates at the desired frequency. Reads from this register return the current clock frequency to provide feedback to the operating system.

Dynamic voltage scaling implemented over a complete processor system chip-set. Not only is the voltage and clock frequency dynamically varied on the microprocessor chip, but on the external SRAM memory chips, on the I/O interface chip, and on the external processor system bus as well.

7.11.1 Comparison to Other ARM Processors

Another goal of this work was to see how much the intrinsic energy-efficiency of a microprocessor could be improved without the benefit of DVS. A key benefit of implementing a commercial ARM8 processor core was that the prototype processor

could be compared against other commercial ARM microprocessor implementations. One of these implementation is the StrongArm SA-110, which is the most energy-efficient commercial microprocessor available to date.

Energy consumption, in capacitance/cycle to normalize out the dependence on V_{DD}, is plotted for the prototype processor and four commercial ARM microprocessors in Figure 7.50. In addition, the processors performance and process technology is given. The capacitance/cycle is broken out to show that which is consumed by the core, and that which is consumed by the cache and the rest of the chip.

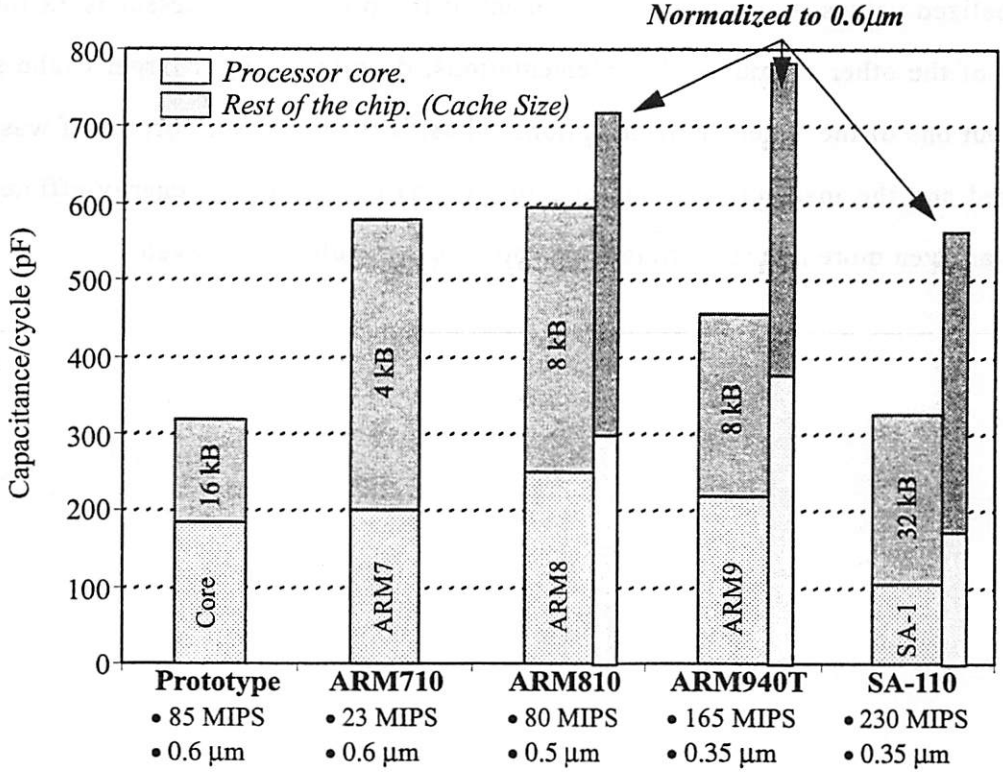


FIGURE 7.50 : Capacitance/cycle of Various ARM Microprocessors.

The prototype processor has the lowest capacitance/cycle of the five implementations, with the SA-110 a close second. However, since the ARM810, ARM940T, and SA-110 have the benefit of a better CMOS process technology, the capacitance/cycle for these three were normalized to the 0.6μm process technology of the prototype processor. Compared against the normalized values, the prototype

processor demonstrates almost 2x lower capacitance/cycle than any of the four commercial processors, validating the energy-efficient design methodology presented in this work. Despite the microarchitectural constraint of using the ARM8 processor core, the prototype system was still able to demonstrate a significant reduction in capacitance/cycle.

Since the cache sub-system was designed in its entirety with energy-efficiency in mind, it is interesting to see how the non-core component of the prototype processor's capacitance/cycle compares against the other implementations. Comparing the normalized values, the non-core component of the prototype processor is 3x lower than any of the other commercial implementations, despite having a larger cache size than all but one of the other implementations. Thus, if the processor core itself was re-architected and the instruction-set architecture (ISA) designed with energy-efficiency in mind, an even more energy-efficient microprocessor could be achieved.

8 Conclusions

Processor systems are widely prevalent in portable devices, which demand increasingly higher levels of energy-efficiency. Processor energy-efficiency has lagged behind custom ASICs and DSP chips, such that while the processor carries only a fraction of the device's computation load, it is a significant, if not dominant, component of the overall system energy consumption. This thesis has demonstrated both design techniques, and a design methodology, to significantly improve processor energy-efficiency to enable smaller, more powerful, and longer running portable devices.

Dynamic voltage scaling has demonstrated to be the most significant design technique, providing an increase in energy-efficiency in excess of 10x. While DVS requires modifications to both the circuit design and design flow, which diminish energy-efficiency by 10-20%, this reduction is overwhelming compensated by the 10x increase.

Furthermore, quantitative energy-efficiency metrics have enabled an energy-efficient design methodology which has provided a further increase of 2-5x in energy-efficiency. This is achieved by optimizing both performance and energy consumption at all levels of the design hierarchy, as opposed to a more traditional design approach which relegates energy consumption to a secondary concern. A prototype system has successfully validated the design techniques and methodology presented in this work.

8.1 Summary of Research Contributions

The goal of this research is to significantly improve processor system energy-efficiency by combining the lessons learned in low-power DSP design with the unique design constraints of a general-purpose processor to develop a new, more energy-efficient, processor design methodology. Several key research contributions which addresses this goal are:

- Developed the technique of Dynamic Voltage Scaling (DVS) for a general-purpose microprocessor to adaptively vary the processor's supply voltage and clock frequency, under operating system control. This allows the processor to provide high performance when required, while minimizing energy consumption during the remaining low-performance periods of time.
- Developed an energy-conscious design flow which enables energy consumption optimization at all levels of the design flow, including the high-level C behavioral simulator, where optimizations can have the biggest impact on energy-efficiency. The new flow also eliminates the extra complexity added by DVS to a more traditional design flow.
- Developed an energy-efficient architectural design methodology for all aspects of a processor system, including system-level optimizations, as well as optimizations targeted for the processor core and cache system.
- Developed an energy-efficient circuit design methodology for all aspects of digital circuit design, while meeting the circuit design constraints imposed by DVS.
- Demonstrated the above concepts by implementing a prototype processor system, consisting of four custom chips in a 0.6μm CMOS process technology, that can operate over the range of 1.2-3.8V, 5-80 MHz, and 0.54-5.6 mW/MIP. Through DVS, the system can deliver a peak performance of 85 Dhrystone 2.1 MIPS, with an average power dissipation as low as 3.24mW. This yields as much as 26,000

MIPS/W, which is more than 10x higher than the most energy-efficient microprocessor currently available.

8.2 Current Industry Directions

In the rapidly evolving processor industry, some of the techniques described in this thesis are beginning to come to fruition. Of particular interest is run-time voltage/ frequency adaptation, which was not even considered feasible three or four years ago within the industry, and yet, is rapidly emerging in a variety of products technologies:

- In 1999, Intel introduced SpeedStep, which runs the processor at two different voltages and frequencies, depending upon whether the notebook computer was plugged into an AC outlet, or running of its internal battery.
- In 2000, Transmeta introduced LongRun, which dynamically varies voltage and frequency over the range of 1.2-1.6V and 500-700MHz, providing a 1.8x variation in processor energy consumption. Control of the voltage/frequency is in firmware, which monitors the amount of time the operating system is sleeping.
- In 2000, AMD introduced PowerNow!, which dynamically varies voltage and frequency over the range of 1.4-1.8V and 200-500MHz, providing a 1.7x variation in processor energy consumption. Control of the voltage/frequency is implemented via a software driver which monitors the operating system's measure of CPU utilization.
- In 2001, Intel will introduce the XScale processor, which is essentially the second generation StrongArm. It can dynamically operate over the voltage and frequency range of 0.7-1.75V and 150-800MHz, providing a 6.3x variation in processor energy consumption, the most aggressive range announced to date. Details of the control have not yet been released. By further advancing the energy-efficiency of the original StrongArm, this device will be able to deliver 1000 MIPS with an

average power dissipation as low as 50mW at 0.7V, yielding an effective MIPS/Watt as high as 20,000.

8.3 Future Research Directions

This thesis has provided the groundwork for a variety of continuing research directions. Further research is required on dynamic voltage scaling, as well as all aspects of energy-efficient design.

Integrating the voltage converter onto the same chip as the processor could be explored. Integration would enable further research on multiple, variable voltage supplies, without adversely impacting system cost, size, and complexity. One potential use of an additional supply would be for the external processor bus, which could then operate at a speed independent of the processor core. This would allow high-speed DMA to the main memory, so that even when the processor core is operating at low speed, high-bandwidth I/O-memory transactions could still occur. Additional research areas would be applying DVS to external I/O devices, such as a radio, which could likewise dynamically trade-off performance (bandwidth) versus energy consumption.

Another research direction would the further exploration of instruction set architecture and microarchitecture for improving energy-efficiency. Of particular interest are VLIW and parallel processor architectures which explicitly expose their parallelism, and do not suffer from exponentially increasing energy consumption with parallelism as is the case with superpipelined and superscalar processor architectures.

As process technology continues to advance, energy consumed by interconnect will consume an increasingly larger fraction of the total energy consumption. Thus, further investigation of low-swing interconnects could yield additional improvement of processor system energy-efficiency.

References

[aebi97] D. Aebischer, et. al., "A 2.1-MHz Crystal Oscillator Time Base with a Current Consumption under 500nA", *IEEE Journal of Solid State Circuits*, Vol. 32, No. 7, Jul. 1997, pp. 999-1005.

[arm94 Advanced RISC Machines, Ltd., *ARM710 Data Sheet*, Technical Document, Dec. 1994.

[arm95] Advanced RISC Machines, Ltd., *Introduction to Thumb*, Developer Technical Document, Mar. 1995.

[arm96a] Advanced RISC Machines, Ltd., *ARM Architecture and Implementation Reference*, Document Number ARM-DDI-0100A-I, Feb. 1996.

[arm96b] Advanced RISC Machines, Ltd., *ARM 8 Data Sheet*, Document Number ARM-DDI-0100A-I, Feb. 1996.

[arm97] Advanced RISC Machines, Ltd., *ARM Software Development Toolkit Reference Guide*, Document Number ARM-DUI-0041A, Jan. 1997.

[bund93a] J. Bunda, et. al., "16-Bit vs. 32-Bit Instructions for Pipelined Architectures", *Proceedings of the 20th International Symposium on Computer Architecture*, May 1993, pp. 237-46.

[bund93b] J. Bunda, *Instruction-Processing Optimization Techniques for VLSI Microprocessors*, Ph.D. Thesis, The University of Texas at Austin, 1993.

[bund94] J. Bunda, W.C. Athas, and D. Fussell, "Evaluating Power Implications of CMOS Microprocessor Design Decisions", *Proceedings of the 1994 International Workshop on Low-Power Design*, Napa Valley, CA, April 1994.

[burd94] T. Burd, *Low-Power CMOS Library Design Methodology*, M.S. Thesis, University of California, Berkeley, Document No. UCB/ERL M94/89, 1994.

[burd94b] T. Burd, B. Peters, *A Power Analysis of a Microprocessor: A Study of an Implementation of the MIPS 3000 Architecture*, ERL Technical Report, University of California, Berkeley, 1994.

[burn97] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*, 2nd Edition, Addison-Wesley, Reading, MA, 1997.

[burs97] A. Burstein, *Speech Recognition for Portable Multimedia Terminals*, Ph.D. Thesis, University of California, Berkeley, Document No. UCB/ERL M97/14, 1997.

[chan92] A. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low-Power CMOS Digital Design", *IEEE Journal of Solid State Circuits*, Apr. 1992, pp. 473-84.

[chan94] A. Chandrakasan, A. Burstein, and R.W. Brodersen, "A Low Power Chipset for Portable Multimedia Applications", *IEEE Journal of Solid State Circuits*, Vol. 29, No. 12, Dec. 1994, pp. 1415-28.

[chan95] A. Chandrakasan, R.W. Brodersen, *Low-power Digital CMOS Design*, Kluwer Academic Publishers, Boston, 1995.

[chan96] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data Driven Signal Processing: An Approach for Energy Efficient Computing", *Proceedings of the 1996 International Workshop on Low-Power Design*, Aug. 1996, pp. 347-52.

[culb94] M. Culbert, "Low Power Hardware for a High Performance PDA", *Proceedings of the Thirty-Ninth IEEE Computer Society International Conference*, Mar. 1994, pp. 144-7.

[de99] V. De and S. Borkar, "Technology and Design Challenges for Low Power and High Performance", *Proceedings of the IEEE Symposium on Low Power Electronics and Design*, Aug. 1999, pp. 163-8.

[dec98] Digital Equipment Corporation, *DIGITAL Semiconductor SA-1100 Microprocessor Technical Reference Manual*, Document EC-R5MTB-TE, Jan 1998.

[endo96] Y. Endo, et. al., "Using Latency to Evaluate Interactive System Performance", *Proceedings of Operating Systems Design and Implementation*, Nov. 1996.

[free94] P. Freet, "The SH Microprocessor: 16-Bit Fixed Length Instruction Set Provides Better Power and Die Size", *Proceedings of the Thirty-Ninth IEEE Computer Society International Conference*, Mar. 1994, pp. 486-8.

[mull86] R. Muller, T. Kamins, *Device Electronics for Integrated Circuits*, Wiley, New York, 1986.

[gary94] S. Gary, et. al., "The PowerPC 603 Microprocessor: A Low-Power Design for Portable Applications", *Proceedings of the Thirty-Ninth IEEE Computer Society International Conference*, Mar. 1994, pp. 307-15.

[gec94] GEC Plessey Semiconductor, *ARM60 Data Sheet*, Technical Document, Aug 1994.

[gonz95] R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Processors", *Proceedings of the IEEE Symposium on Low Power Electronics*, Oct. 1995, pp. 12-3.

[gray93] P. Gray, R. Meyer, *Analysis and Design of Analog Integrated Circuits*, Wiley, New York, 1993.

[henn95] J. Hennessy, D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, 1995.

[ho99] R. Ho, K. Mai, M. Horowitz, "Scaling Implications for CAD", *Proceedings of the IEEE International Conference for Computer-Aided Design*, Nov. 1999.

[hp95] Hewlett Packard, *CMOS 14TA/B Reference Manual*, Document Number #A-5960-7127-3, Jan. 1995.

[horo94] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-Power Digital Design", *Proceedings of the IEEE Symposium on Low Power Electronics*, Oct. 1994, pp. 8-11.

[huan93] J. Huang, et. al., "A Robust Physical and Predictive Model for Deep-Submicrometer MOS Circuit Simulation", *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1993, pp. 14.2.1-4.

[idt95] Integrated Device Technology, Inc., *Enhanced Orion 64-Bit RISC Microprocessor*, Data Sheet, Sep. 1995.

[iked95] T. Ikeda, "ThinkPad Low-Power Evolution", *Proceedings of the IEEE Symposium on Low Power Electronics*, Oct. 1995, pp. 6-7.

[inte95] Intel Corp., *Embedded Ultra-Low Power Intel486TM GX Processor*, SmartDieTM Product Specification, Dec. 1995.

[john90] M. Johnson, *Superscalar Microprocessor Design*, Englewood, NJ: Prentice Hall, 1990.

[joup89] N. Jouppi and D. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines", *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 1989, pp 272-82.

[kaen90] V. von Kaenal, P. Macken, and M. Degrauwe, "A Voltage Reduction Technique for Battery-operated Systems", *IEEE Journal of Solid State Circuits*, Vol. 25, No. 10, Oct. 1990, pp. 1136-40.

[kawa98] S. Kawashima, et. al., "A Charge-Transfer Amplifier and an Encoded-Bus Architecture for Low-Power SRAM's", *IEEE Journal of Solid State Circuits*, Vol. 33, No. 5, May 1998, pp. 793-9.

[kuni95] S. Kunii, "Means of Realizing Long Battery Life in Portable PCs", *Proceedings of the IEEE Symposium on Low Power Electronics*, Oct. 1995, pp. 20-3.

[kuro96] T. Kuroda, et. al., "A 0.9-V, 150-MHz, 10-mW, 4 mm^2, 2-D Discrete Cosine Transform Core Processor with Variable Threshold-Voltage (VT) Scheme", *IEEE Journal of Solid State Circuits*, Vol. 31, No. 11, Nov. 1996, pp. 1770-9.

[kuro98] T. Kuroda, et. al., "Variable Supply-voltage Scheme for Low-power High-speed CMOS Digital Design", *IEEE Journal of Solid State Circuits*, Vol. 33, No. 3, Mar. 1998, pp. 454-62.

[land94] P. Landman, J. Rabaey, "Black-Box Capacitance Models for Architectural Power Analysis", *Proceedings of the 1994 International Workshop on Low-Power Design*, Napa Valley, CA, April 1994.

[lee97] W. Lee, et. al., "A 1-V Programmable DSP for Wireless Communications", *IEEE Journal of Solid State Circuits*, Vol. 32, No. 11, Nov. 1997, pp. 1766-76.

[liu73] C. Liu and J. Layland, "Scheduling Algorithms for Multi-Programming in a Hard Real-time Environment", *Proceedings of CACM 20*, 1973.

[lown93] P. Lowney, et. al., "The Multiflow Trace Scheduling Compiler", *The Journal of Supercomputing*, Vol. 7, Boston: Kluwer Academic Publishers, 1993, pp. 51-142.

[mark00] D. Markovic, *Analysis and Design of Low-Energy Clocked Storage Elements*, M.S. Thesis, University of California, Berkeley, Document No. UCB/ERL M00/64, 2000.

[mont96] J. Montanaro, et. al., "A 160-MHz 32-b 0.5-W CMOS RISC Microprocessor", *IEEE Journal of Solid State Circuits*, Vol. 31, No. 11, Nov. 1996, pp. 1703-14.

[mont96b] J. Montanaro, et. al., "A 160MHz 32b 0.5W CMOS RISC Microprocessor", *Proceedings of the Thirty-Ninth IEEE International Solid-State Circuits Conference - Slide Supplement*, Feb. 1996, pp. 170-1.

[muto95] S. Mutoh, et. al., "1-V Power Supply High-Speed Digital Circuit Technology with Multithreshold-Voltage CMOS", *IEEE Journal of Solid State Circuits*, Vol. 30, No. 8, Aug. 1995, pp. 847-54.

[niel94] L. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, "Low-power Operation Using Self-timed Circuits and Adaptive Scaling of the Supply Voltage", *IEEE Transactions on VLSI Systems*, Vol. 2, No. 4, Dec. 1994.

[peri00] T. Pering, *Energy-Efficient Operating System Techniques*, Ph.D. Thesis, University of California, Berkeley, 2000.

[pier96] R. Pierret, *Semiconductor Device Fundamentals*, Addison Wesley, Reading, MA, 1996.

[raba96] J. Rabaey, *Digital Integrated Circuits, A Design Perspective*, Prentice Hall, Upper Saddle River, NJ, 1996.

[shir96] T. Shiraishi, et. al., "A 1.8V 36mW DSP for the Half-rate Speech CODEC", *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 1996, pp. 371-4.

[smit89] M. Smith, M. Johnson, and M. Horowitz, "Limits on Multiple Issue Instruction", *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 1989, pp 290-302.

[spec94] Standard Performance Evaluation Corporation, *SPEC Run and Reporting Rules for CPU95 Suites*, Technical Document, Sep. 1994.

[stra94] A. Stratakos, S. Sanders, and R.W. Brodersen, "A Low-voltage CMOS DC-DC Converter for Portable Battery-operated Systems", *Proceedings of the Twenty-Fifth IEEE Power Electronics Specialist Conference*, June 1994, pp. 619-626.

[stra98] A. Stratakos, *High-Efficiency, Low-Voltage DC-DC Conversion for Portable Applications*, Ph.D. Thesis, University of California, Berkeley, 1998.

[su95] C. Su, A. Despain, "Cache Designs for Energy Efficiency", *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences*, Jan. 1995, pp. 306-315.

[sylv98] D. Sylvester and K. Keutzer, "Getting to the bottom of deep submicron", *Proceedings of the International Conference on CAD*, 1998, pp. 203-211.

[sze81] S. Sze, *Physics of Semiconductor Devices*, Wiley, New York, 1981.

[tiwa96] V. Tiwari, et. al., "Instruction Level Power Analysis and Optimization of Software", *Journal of VLSI Signal Processing*, Vol. 13, Nos. 2/3, Aug/Sep 1996, pp. 223-238.

[toh88] K. Toh, P. Ko, R. Meyer, "An Engineering Model for Short-Channel MOS Devices", *IEEE Journal of Solid-State Circuits*, Vol. 23, No. 4, April 1988.

[tser96] E. Tsern, and T. Meng, "A Low Power Video-rate Pyramid VQ Decoder", *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 11, Nov. 1996, pp. 1789-94.

[ueda93] K. Ueda, et. al., "A 16b Low-power-consumption Digital Signal Processor", *Proceedings of the IEEE International Solid-State Circuits Conference*, San Francisco, Feb. 1993, pp. 28-9.

[veen84] H. Veendrick, "Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits", *IEEE Journal of Solid State Circuits*, Vol. 19, No. 4, August 1984.

[vitt80] E. Vittoz, "Micropower IC", *Proceedings of the IEEE European Solid-State Circuits Conference*, Sep. 1980, pp. 174-89.

[wall93] D. Wall, *Limits of Instruction-Level Parallelism*, DEC WRL Research Report 93/6, Nov. 1993.

[wei00] G. Wei, et. al., "A Variable-frequency Parallel I/O Interface with Adaptive Power Supply Regulation", *Proceedings of the IEEE International Solid-State Circuits Conference*, San Francisco, Feb. 2000, pp. 298-9.

[weic84] R. Weicker, "Dhrystone: A Synthetic Systems Programming Benchmark", *Communications of the ACM*, Vol. 27, No. 10, Oct. 1984, pp. 1013-30

[west93] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, Reading, MA, 1993.

[yuan89] J. Yuan, C. Svensson, "High-Speed CMOS Circuit Techniques", *IEEE Journal of Solid-State Circuits*, Vol. 24, No. 1, Feb. 1989

[zhan00] H. Zhang, et. al., "A 1-V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing", *IEEE Journal of Solid-State Circuits*, Vol. 35, No. 11, Nov. 2000, pp. 1697-1704.