Power Reduction of Digital Signal Processing Systems using Subthreshold Operation

Michael B. Henry

A thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Masters of Science
in
Computer Engineering

Masters Committee:
Leyla Nazhandali, Chair
A. A. (Louis) Beex
Cameron Patterson

May 12, 2009
Blacksburg, Virginia

Keywords:
Low Power, Digital Signal Processing,
Subthreshold Operation, Minimum Energy

Power Reduction of Digital Signal Processing Systems using Subthreshold Operation

Michael B. Henry

(ABSTRACT)

Over the past couple decades, the capabilities of battery-powered electronics has expanded dramatically. What started out as large bulky 2-way radios, wristwatches, and simple pacemakers, has evolved into pocket sized smart-phones, digital cameras, person digital assistants, and implantable biomedical chips that can restore hearing and prevent heart attacks. With this increase in complexity comes an increase in the amount of processing, which runs on a limited energy source such as a battery or scavenged energy. It is therefore desirable to make the hardware as energy efficient as possible. Many battery-powered systems require digital signal processing, which often makes up a large portion of the total energy consumption. The digital signal processing of a battery-powered system is therefore a good target for power reduction techniques. One method of reducing the power consumption of digital signal processing is to operate the circuit in the subthreshold region, where the supply voltage is lower than the threshold voltage of the transistors. Subthreshold operation greatly reduces the power and energy consumption, but also decreases the maximum operating frequency. Many digital signal processing applications have real-time throughput requirements, so various architectural level techniques, such as pipelining and parallelism, must be used in order to achieve the required performance.

This thesis investigates the use of parallelization and subthreshold operation to lower the power consumption of digital signal processing applications, while still meeting throughput requirements. Using an off the shelf fast fourier transform architecture, it will be shown that through parallelization and subthreshold operation, a 70 % reduction in power consumption can be achieved, all while matching the performance of a nominal voltage single core architecture. Even better results can be obtained when an architecture is specifically designed for subthreshold operation. A novel Discrete Wavelet Transform architecture is presented that is designed to eliminate the need for memory banks, and a power reduction of 26x is achieved compared to a reference nominal voltage architecture that uses memory banks. Issues such as serial to parallel data distribution, dynamic throughput scaling, and memory usage are also explored in this thesis. Finally, voltage scaling greatly increases the design space, so power and timing analysis can be very slow due long SPICE simulation times. A simulation framework is presented that can characterize subthreshold circuits accurately using only fast gate level design automation tools.

Acknowledgements

First of all, I would like to thank my family for their continued encouragement and the occasional financial support.

Thanks to Dr. Leyla Nazhandali for her continued support and for providing me with very interesting research topics.

Thanks to Dr. A. A. Beex and Dr. Cameron Patterson for their participation in my thesis committee.

Finally, thanks to Syed Haider, Steven Griffin, and Vignesh Vivekraja for their assistance with the work presented in this thesis.

Contents

List of Tables

List of Figures

List of Abbreviations

ASIC	Application Specific Integrated Circuit
BP	Butterfly Processor (FFT architecture)
BSIM	Berkeley Short-channel Insulated Gate Field Effect Transistor Model
C_L	Load Capacitance
CMOS	Complementary Metal Oxide Semiconductor
DSP	Digital Signal Processing
DWT	Discrete Wavelet Transform
EDA	Electronic Design Automation
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
JPEG	Joint Photographic Experts Group
JPEG2000	Joint Photograph Experts Group (2000 Revision)
NAND	Negated And
NMOS	N-type Metal Oxide Semiconductor
NPC	Non-Performance Constrained
PC	Performance Constrained
PE	Processing Element (FFT architecture)
PMOS	P-type Metal Oxide Semiconductor
RAM	Random Access Memory
ROM	Read Only Memory
SPICE	Simulation Program with Integrated Circuit Emphasis
SRAM	Static random Access Memory
V_{dd}	Supply Voltage
V_{th}	Threshold Voltage
WT	Wavelet Transform

1 Introduction

1.1 Low Power Digital Signal Processing

Over the past couple of decades, the capabilities of battery-powered electronics has expanded dramatically. What started out as large bulky 2-way radios, wristwatches, and simple pacemakers, has evolved into pocket sized smart-phones, digital cameras, PDAs, and implantable biomedical chips that can restore hearing and prevent heart attacks. In battery-powered systems, the hardware has tight power and energy constraints, due to the desire to extend the battery life as much as possible. For example, consider an implantable biomedical device that requires surgery to change the battery every 5 years. If a hardware designer can apply power saving techniques and double the battery life, then the patient would only have to undergo potentially dangerous surgery half as many times. Power consumption is also important in applications that must scavenge energy from the environment. If a designer were to cut the power consumption of a communications satellite in half, then the size of the solar wings could be reduced by half. This would reduce the payload size and greatly reduce costs.

One common element in battery-powered applications is that they are data-intensive and require a large amount of digital signal processing (DSP). Often the DSP is a major source of power consumption, so it is a good target for power reduction techniques. Many DSP applications have real-time throughput requirements that must be met however, so power reduction techniques must be used carefully to ensure that the hardware is capable of meeting the throughput requirements. These *Performance Constrained* (PC) DSP applications include digital baseband processing sys-

tems, portable imaging and surveillance systems, and communication satellites, which have both real-time processing requirements and limited battery energy available. In other applications, power consumption is very important, and there is no constraint on the speed of the processing, so performance can be sacrificed for lower power. These so called *Non-Performance Constrained* (NPC) DSP applications include wireless sensor networks, implantable biomedical devices, and interplanetary space applications. A common distinction in these applications is the difficulty in replacing the battery or the unpredictability of scavenging energy from the environment. Another distinction is that, with little to no performance requirement, energy consumption per operation becomes much more important than power consumption. Figure 1.1 provides a summary of both PC and NPC DSP applications.

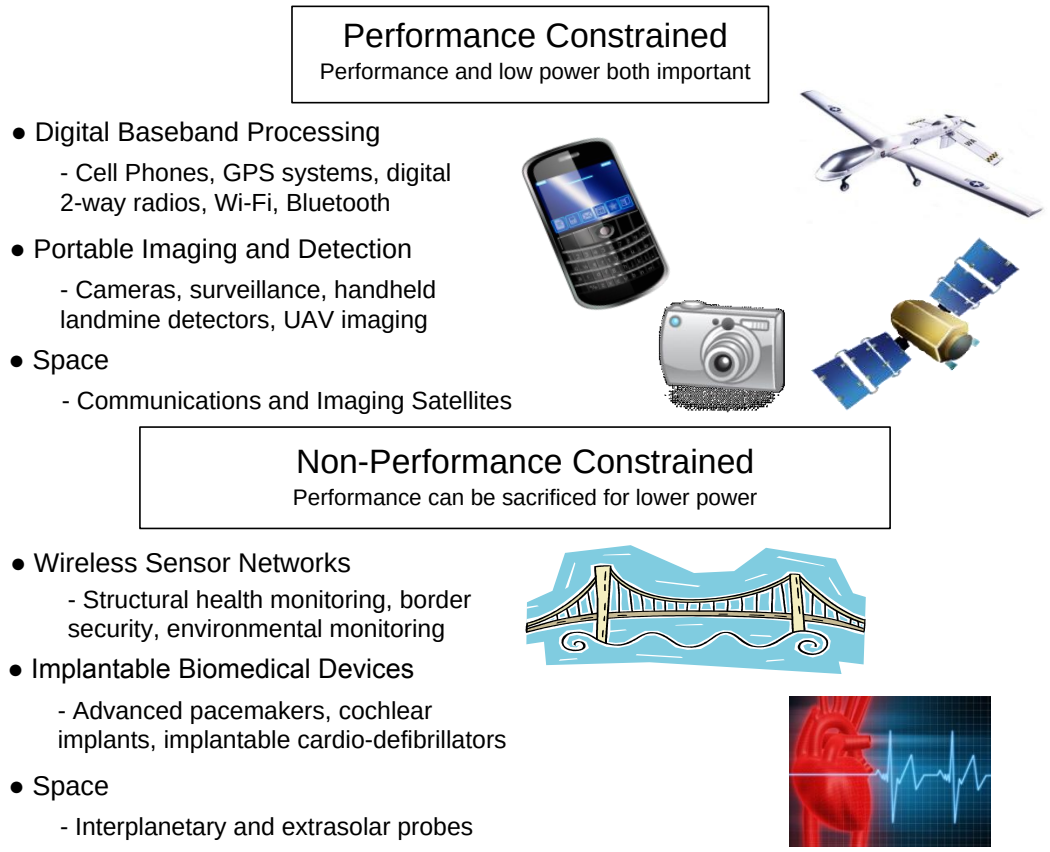


Figure 1.1: Performance constrained (PC) and non-performance constrained (NPC) DSP applications.

There is a variety of methods available to reduce the energy and power consumption of a digital signal processing system. Many approaches focus on circuit and application specific techniques, such as reducing the amount of required processing by reducing precision, throughput, or bit rate [17], offloading computation to devices with more relaxed power constraints [5], or tweaking the arithmetic and control hardware to take advantage of application specific conditions. There are also techniques for power reduction that are non-application specific, including widely popular techniques such as voltage and frequency scaling and leakage power reduction [6]. Voltage scaling combined with frequency scaling is especially popular because reducing the supply voltage of a CMOS circuit reduces the active and leakage power polynomially, while only reducing the frequency linearly.

Traditionally, the limit to voltage scaling has been about half of the nominal voltage, due to incorrect operation of certain components like SRAM and phased-locked loops at lower voltages, an increased sensitivity to process variation, and a large slowdown of the circuit at lower voltages. Recently, however, it has been shown that using custom digital components designed to operate at very low voltages, it is possible to operate a CMOS circuit at a supply voltage that is lower than the threshold voltage of the transistors, i.e. in the *subthreshold region*. As was previously said, the circuit is more sensitive to process variation and is very slow in the subthreshold region, but the energy reduction is extremely large. For NPC applications with little to no performance constraints, the process variation and speed reduction can be tolerable. For PC applications with tighter performance requirements, subthreshold operation in combination with architectural level techniques such as pipelining and parallelization can be used to both reduce power consumption, and meet throughput requirements.

This thesis investigates the use of parallelization and subthreshold operation to lower the power consumption of DSP applications, while still meeting throughput requirements. Using an off the shelf FFT architecture, it will be shown that through parallelization and subthreshold operation, a 70 % reduction in power consumption can be achieved, all while matching the performance of a nominal voltage single core architecture. Even better results can be obtained when an architecture

is specifically designed for subthreshold operation. A novel Discrete Wavelet Transform architecture is presented that is designed to eliminate the need for SRAM banks, and a power reduction of 26x is achieved compared to a reference nominal voltage architecture. Issues such as serial to parallel data distribution, dynamic throughput scaling, and memory usage are also explored in this thesis. Finally, voltage scaling greatly increases the design space and power and timing analysis can be very slow due to long SPICE simulation times. A simulation framework is presented that can characterize subthreshold circuits accurately using only fast gate-level EDA tools.

1.2 Related Work

As was mentioned in the previous section, some power reduction techniques are application specific and some are non-application specific. The techniques and methodologies in this thesis are non-application specific and can be applied to most digital signal processing systems, so with that in mind, the following are recent related works that investigate non-application specific methods for reducing the power and energy consumption of digital signal processing. Specific attention is paid to works that operate in the subthreshold region.

- [6] includes the most popular techniques for reducing the power consumption of CMOS digital circuits, including voltage scaling, clock frequency scaling, parallelizing to reduce power, algorithm analysis, etc. This book serves as a nice foundation for the work presented in this thesis; however, the book predates the introduction of subthreshold operation.

- [24] presents a fabricated 1024 pt FFT processor that operates in the subthreshold region. The minimal operating voltage is 180 mV, where it consumes 90 nW with a clock speed of 164 Hz. The optimal energy operating point is 350 mV with a clock frequency of 9.6 kHz, where it consumes 600 nW.

- [20] explores the effect of parallelization and pipelining on the power consumption and energy consumption of an FIR filter, with an emphasis on subthreshold operation. Some of these

techniques are used in the case studies and in Chapter 6, where these ideas are elaborated on as potential future research.

- [18] presents a fabricated ultra low power sensor network processor that operates in the subthreshold region, and analyzes the power and energy consumption across a wide range of applications, including digital signal processing techniques such as FIR filtering and convolution. This is a more general purpose processor, compared to the ASICs detailed in this thesis, but the concepts are still relevant.

- [11] investigates chips operating in the near-threshold and subthreshold region. Specifically, it looks at memory and cache issues as well as multi-core general purpose processors. In situations where an ASIC is not appropriate, but the power reduction benefits of near and subthreshold operation are needed, [11] provides relevant analysis.

1.3 Contributions

The contributions of this thesis are as follows:

- In Chapter 2, a background on subthreshold voltage technology is presented. The theory of operation in this region as well as the trade-offs are explored.

- The simulation of subthreshold and voltage scaled circuits can be slow due to increased computation times and a high accuracy requirement. In addition, EDA tools are not setup to analyze circuits at subthreshold voltages. In Chapter 3, a simulation framework is presented that can quickly analyze the performance and power of a circuit across a wide range of voltages using only fast gate level tools. This work is published in the proceedings of the ISCAS 2009 conference [13].

- In Chapter 4, a novel Discrete Wavelet Transform (DWT) is presented that is optimized for subthreshold operation. The architecture is parallelized to deliver the performance of a circuit at nominal voltage, while delivering the power savings of subthreshold operation. The architecture is designed to eliminate the need for SRAM banks, and a power reduction of 26x is achieved compared to a reference nominal voltage architecture. This work is published in the proceedings of the CASES 2008 conference [14].

- In Chapter 5, a case study of a Fast Fourier Transform (FFT) architecture is explored. The FFT architecture is also parallelized to match the performance at nominal voltage. Compared to a nominal voltage single core architecture, the parallelized subthreshold architecture consumes 70 % less power. In addition, a novel real-time throughput scaling technique is explored that can reduce the power consumption when the performance requirements are relaxed. This method is shown to be much more effective than clock scaling or voltage scaling. This work is published in the proceedings of the HiPEAC 2009 conference [15], and is invited to be published in the 2009 HiPEAC Transactions journal.

- In Chapter 6, future research topics are presented that would build on the work presented in this thesis. Specifically, the future role of subthreshold technology is examined at the micro-architectural level and the system level.

2 Subthreshold Voltage Technology

2.1 Energy vs. Power

The target applications of this thesis all run on batteries with a fixed amount of energy (or scavenged energy from the environment), so hardware must be developed that minimizes the energy consumption of the digital signal processing and therefore prolongs battery life. With this in mind, first an important distinction must be made between energy consumption and power consumption. Energy, measured in Joules, is a measure of the amount of work done, so it is frequently given in terms of energy per operation (i.e. 10 nJ per multiplication) or energy per clock cycle. Power consumption, on the other hand, is a measure of energy divided by time. In NPC applications, the clock cycle is of little concern to the developer, so it is more important to measure the energy per operation. This is especially important with two functionally equivalent designs with different clock speeds, since less power does not always mean less energy. PC applications on the other hand, usually have some target clock frequency or throughput. This means that power consumption and energy consumption are tied together between two functionally equivalent designs, and power consumption can be used to determine which design consumes the least amount of energy. Therefore, when talking about both PC and NPC applications, the objective is to minimize energy consumption and not power consumption. Power consumption is still a ubiquitous term in digital design though, so in cases where there is a target throughput or frequency, power consumption will be stated.

2.2 Subthreshold Operation

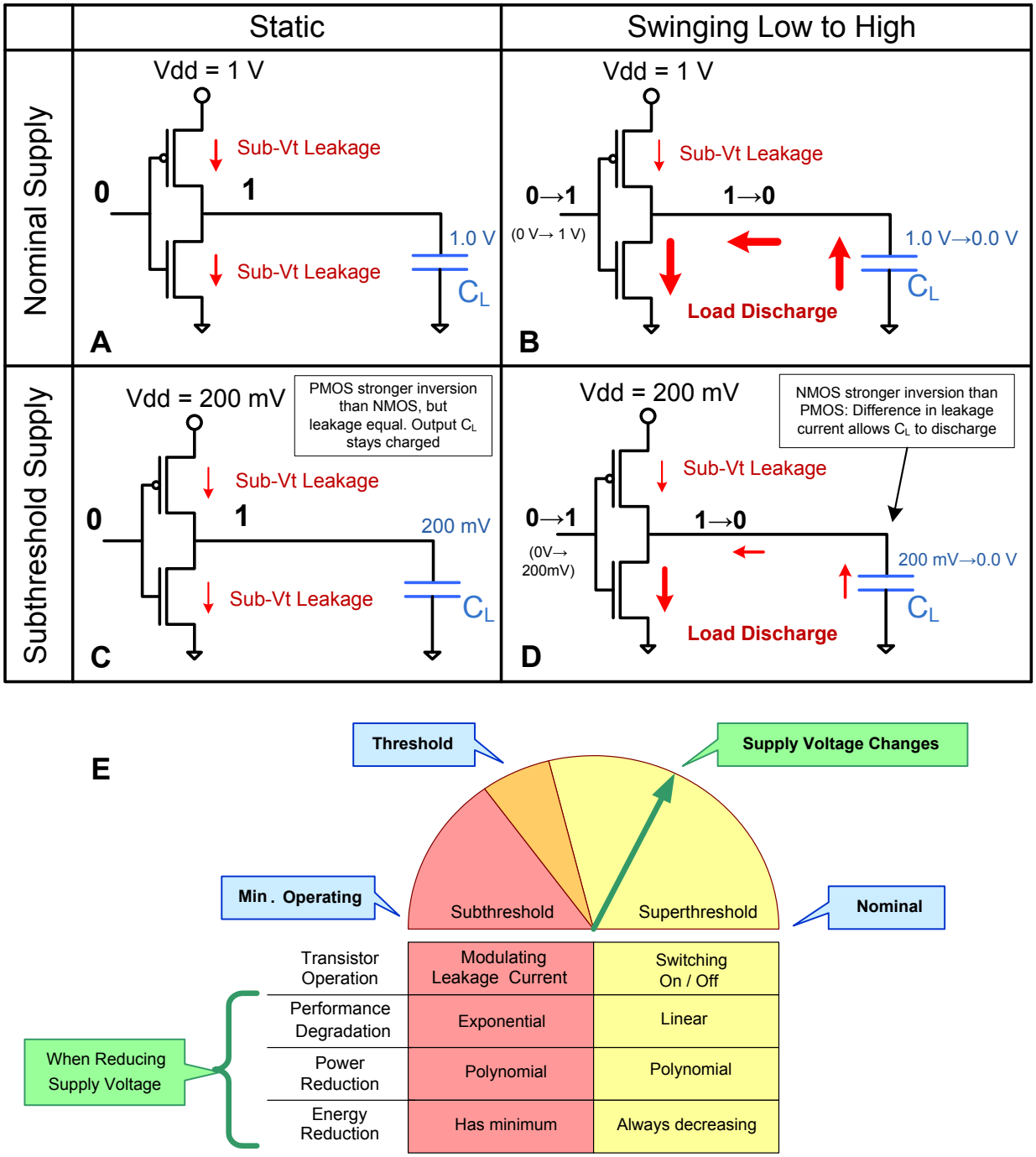


Figure 2.1: Simplified theory of operation of a superthreshold and subthreshold inverter.

As was mentioned in Chapter 1, it is possible to operate a voltage scaled CMOS circuit with a

supply voltage that is less than the threshold voltage of the transistors. Figure 2.1 gives an overview of subthreshold operation by comparing a superthreshold inverter to a subthreshold inverter. In the superthreshold region, when a CMOS circuit is static, there still exists subthreshold leakage current that flows through the PMOS and NMOS networks. This is considered wasted current in the superthreshold region, but in the subthreshold region, this current allows successful operation of a CMOS circuit. Figure 2.1.C shows a static inverter with a low input, high output and a supply voltage of 200 mV (assume this is lower than the threshold voltage). The NMOS and PMOS networks are leaking the same amount of current, but in this state, the PMOS network has a stronger channel inversion than the NMOS network. This means that the output capacitance stays charged at logic high. Figure 2.1.D shows the input of the subthreshold inverter swinging from low to high. Now the NMOS network is capable of leaking more than the PMOS network, and the output capacitance discharges through the NMOS network, sending the circuit to logic low. The same principles apply when the circuit is swung from logic high to logic low: the PMOS network has a stronger inversion than the NMOS network and the output capacitance charges. Thus, correct operation of a CMOS circuit can still be achieved with a subthreshold supply voltage.

In the subthreshold region, the transistors are operating in the cutoff region with a weak inversion, as opposed to the saturated and triode region with superthreshold devices, so the circuits scale differently when the supply voltage is reduced. Figure 2.1.E provides a summary of the effects of reducing the supply voltage on performance and power in both the superthreshold and subthreshold region. For one thing, the performance (maximum frequency) of a CMOS circuit reduces exponentially in the subthreshold region, as opposed to linearly in the superthreshold region. At the same time, power is still reducing polynomially. In terms of energy, reducing the supply voltage and frequency of a circuit in the superthreshold region always reduces the energy per operation, whereas in the subthreshold region, the energy reduction eventually reaches a minimum and actually starts to rise again at a certain subthreshold supply voltage. This effect will be explored in more detail in the next chapter. With these considerations in mind, it becomes clear that operating in the subthreshold region means taking a huge performance penalty. For NPC systems that have

little concern about processing speed, the performance penalty may be tolerable, but for PC systems, additional architectural level techniques such as parallelization will be needed to restore the lost performance.

3 Fast Simulation Framework

3.1 Introduction

Voltage scaling and subthreshold operation has been shown to be very effective for reducing the power consumption of digital circuits, but the design time can greatly increase when voltage scaling is introduced. This is especially true with subthreshold circuits. Gate level design automation tools are effective at measuring circuits at nominal voltage, but are unable to provide timing and power measurements at very low voltages. This means that a designer must use SPICE simulations, but because of the reduced speed of the circuit and the high accuracy required to measure the very small current levels, transistor level simulators such as SPICE take a long time to simulate subthreshold circuits. This is compounded by the fact that a circuit with voltage scaling must be simulated across a range of voltages to determine the most efficient operating point.

In this chapter, a simulation framework is presented that provides fast and accurate analysis of a gate-level design across a wide range of supply voltages, including the subthreshold region. The simulation framework is based on the experimental methodology of [14, 18, 11, 28]. First, a detailed description of the method is presented and important options and concerns are raised. Second, a detailed analysis of the framework is presented in the presence of varying parameters such as wire length, gate input count, etc. The framework is validated by applying it to a 15,000-transistor design. It is shown that the measured result collected from an accurate transistor-level tool, that takes several hours to obtain, is very close to the estimated result, which only takes a few seconds to obtain.

3.2 Simulation Framework

Figure 3.1 presents the simulation framework. The idea is to characterize a ring oscillator in a circuit simulation program such as SPICE across a range of supply voltages, then normalize the resulting curves. By multiplying the normalized curves by the nominal values of another circuit, a voltage scaling characteristic of that circuit can be obtained. A ring oscillator is comprised of an odd numbered chain of complimentary gates with the output connected back to the input. The frequency of oscillation is related to the rise and fall times of the gates as well as the length of the chain. As the supply voltage of the ring oscillator is reduced, the frequency of oscillation and the power consumption are reduced. The supply voltage of a ring oscillator is varied from nominal voltage all the way down to the minimum operating voltage, which is in the subthreshold region. The resulting frequency and power curves are then normalized by dividing the curve by the value at nominal voltage. These normalized curves can then be used to determine how other circuits respond to supply voltage reduction. All that is needed to characterize an arbitrary design is a fast gate-level design analyzer such as PrimeTime that can collect power and delay characteristics of a circuit at nominal voltage. As shown in Figure 3.1, these numbers can then be multiplied by the normalized ring oscillator curves, producing curves that model the frequency and power consumption of a design at any voltage. The rest of this chapter is organized as follows: First, the framework will be validated analytically using basic equations for timing and power consumption of CMOS circuits. Next, the framework will be validated experimentally by comparing a baseline 2-input NAND ring oscillator to oscillators with modified parameters. Finally, the effectiveness of the framework will be tested for larger circuits by comparing the measured timing and power of a 15,000-transistor multiplier with the estimates obtained from the simulation framework.

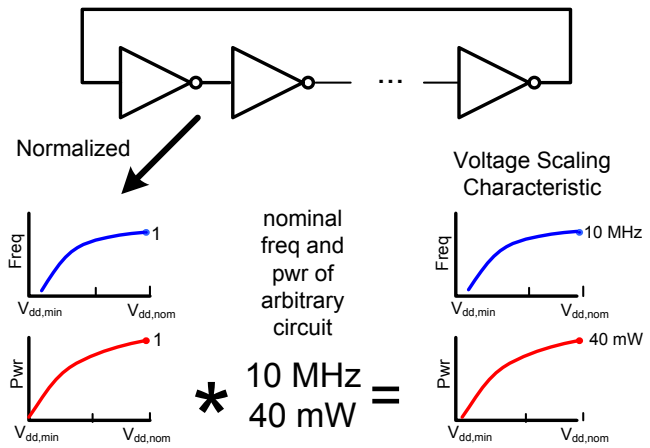


Figure 3.1: Voltage scaling characterization using a normalized ring oscillator.

3.3 Analytical Validation

While a detailed analytical validation is out of the scope of this chapter, it is possible to look at the equations that impact timing and power and gain some insight as to why the normalization approach achieves accurate results. The coefficients of the equations will be broken up into two categories: circuit dependent parameters and circuit independent parameters. Circuit dependent parameters, such as transistor width and load capacitance, vary from circuit to circuit, while circuit independent parameters depend on the technology rather than the circuit. The simulation framework assumes that the normalized curves of two circuits scale the same way. This means that after normalizing timing and power, circuit dependent parameters get canceled out and only circuit independent parameters remain (i.e. a normalized ring oscillator curve is identical to a normalized multiplier curve). It will be shown that certain assumptions have to be made in order for this to be true. When these assumptions don't hold, errors are introduced into the estimate. The extent of the error will be shown in the experimental validation in Section 3.4.

3.3.1 Active Power

Equation (3.1) gives a basic equation for the active power dissipation of a circuit. Equation (3.2) shows Equation (3.1) after normalization to nominal voltage. The $f(V_{dd})$ term implies that the

frequency is dependent of V_{dd}, because as the supply voltage is reduced, the frequency should also be reduced due to the slower propagation speeds.

$$P_{active}(V_{dd}) = C_L V_{dd}^2 f(V_{dd})\alpha \tag{3.1}$$

$$\frac{P_{active}(V_{dd})}{P_{active}(V_{Nom})} = \frac{V_{dd}^2 f(V_{dd})}{V_{Nom}^2 f(V_{Nom})} \tag{3.2}$$

Equation (3.2) shows that normalization cancels out circuit dependent parameters such as the activity factor and load capacitance, meaning the active power of a very large circuit with a low activity factor scales the same as a small circuit that is very active.

3.3.2 Leakage Power

Equation (3.3) shows the equation for leakage, based on the Berkeley short channel field effect model (BSIM), with $V_{gs} = 0$. For details of the coefficients, refer to the BSIM v3.2 documentation [4]. Equation (3.4) shows the equation for leakage current after normalization.

$$I_{ds,off}(V_{dd}) = \mu_o C_{ox} \frac{W}{L} e^{b(V_{dd}-V_{Nom})} v_t^2 (1 - e^{\frac{-V_{dd}}{v_t}}) e^{\frac{-|V_{th}|-V_{off}}{n \cdot v_t}} \tag{3.3}$$

$$\frac{I_{ds,off}(V_{dd})}{I_{ds,off}(V_{Nom})} = \frac{e^{b(V_{dd}-V_{Nom})}(1 - e^{\frac{-V_{dd}}{v_t}})}{(1 - e^{\frac{-V_{Nom}}{v_t}})} \tag{3.4}$$

The b parameter is the drain induced barrier leakage coefficient which can be affected by stacked transistors. Also, Equation (3.3) assumes $V_{ds} = V_{dd}$, which is not the case for stacked transistors. Thus, a different number of stacked transistors will cause a difference between normalized curves. Finally, Equation (3.3) assumes that V_{th} is constant. In reality, V_{th} changes with V_{dd}, W, and L, which is another possible source of error.

3.3.3 Timing

There are many delay models for CMOS circuits, and most simple models rely on an Resistor-Capacitor model, where timing is based on a linear relationship between an effective resistance

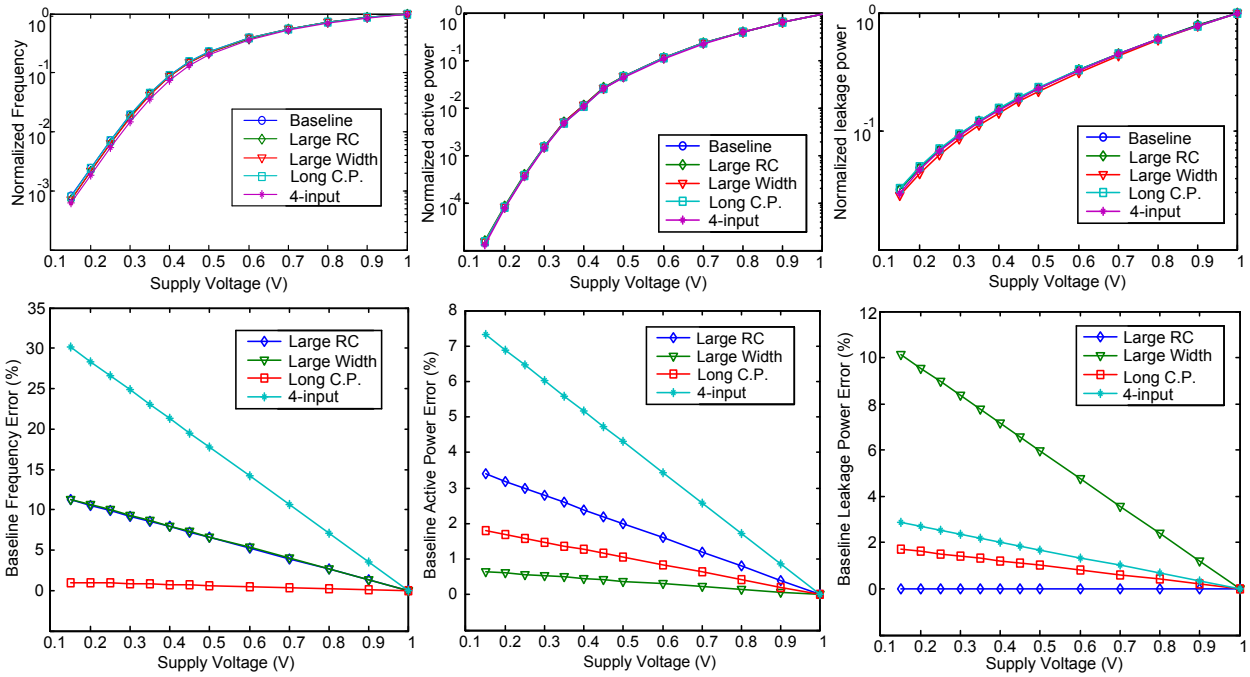


Figure 3.2: Comparing the baseline to normalized ring oscillators with modified circuit parameters.

of the transistor and the load capacitance [25]. Because the relationship is linear, normalization cancels out the load capacitance. The effective resistance is dependent on the drain-source current, which itself depends on the mode of operation of the transistor. Equation (3.4) already showed how normalization cancels out circuit dependent parameters when the transistor is in the cutoff region. Equation (3.5) and Equation (3.6) show the effect of normalization on the saturation current of a transistor.

$$I_{ds,sat}(V_{dd}) = \mu_o C_{ox} \frac{W}{L} (V_{dd} - V_{th})^2 \tag{3.5}$$

$$\frac{I_{ds,sat}(V_{dd})}{I_{ds,sat}(V_{Nom})} = \frac{(V_{dd} - V_{th})^2}{(V_{Nom} - V_{th})^2} \tag{3.6}$$

Equation (3.5) assumes that $V_{gs} = V_{ds} = V_{dd}$. As was seen with the leakage power, due to stacking this is not always the case and will lead to error. Also, the framework once again assumes that threshold voltage is a circuit-independent parameter, whereas in reality, threshold voltage can change with different transistor sizes and supply voltages.

3.4 Experimental Validation

To validate the simulation framework, first baseline normalized ring oscillator curves were obtained from an 11-stage 2-input NAND ring oscillator with no load capacitance or resistance and with transistor lengths and widths taken from a commercial standard cell library. It was simulated in SPICE and the frequency and power consumption were measured from nominal voltage down to the minimum operating voltage. These baseline curves serve as the foundation of the simulation framework and are used to determine the power and timing of other circuits, so they need to be tested to ensure they are accurate. To do so, the baseline is compared to normalized curves of ring oscillators with varied parameters. *If normalized curves of the other ring oscillators match the curves of the baseline, this implies that the circuits scale in the same fashion, and that the curves of the other circuits can be derived from the baseline using only the nominal frequency and power consumption.* These experiments were performed with a 90 nm technology library from a top foundry.

Figure 3.2 shows the baseline curves compared against normalized curves obtained from ring oscillators that were created to test varying load resistance and capacitance, transistor width, critical path length, and gate input count. To simulate a load capacitance and resistance on the output of a gate, a 240 ohm resistor and grounded 100 pF capacitor were placed in between each stage of the baseline ring oscillator. To simulate a varying critical path depth, a ring oscillator with 31 stages was simulated. Another simulation involved setting the width of the transistors to 10x that of the baseline to account for transistor sizing done to optimize circuits. Finally, an 11-stage oscillator comprised of 4-input NAND gates was simulated to determine the error caused by the stacking effect.

Figure 3.2 also shows the error between the baseline curves and the modified curves. The curves were fitted to make the linearity of the error more clear. Only nominal numbers are used in the framework and as the supply voltage gets farther from nominal, the error increases. For timing, varying the critical path length introduces little error, while varying transistor width and large

load RCs introduce a moderate amount of error, considering the modified ring curves represent an extreme width and load capacitance. For both active power and timing, increasing the number of gate inputs introduces a large amount of error. This is most likely due to the stacking effect mentioned in the analytical validation. For leakage power, varying the width introduces a moderate amount of error, which is likely due to the threshold voltage variation introduced by making the transistors wider.

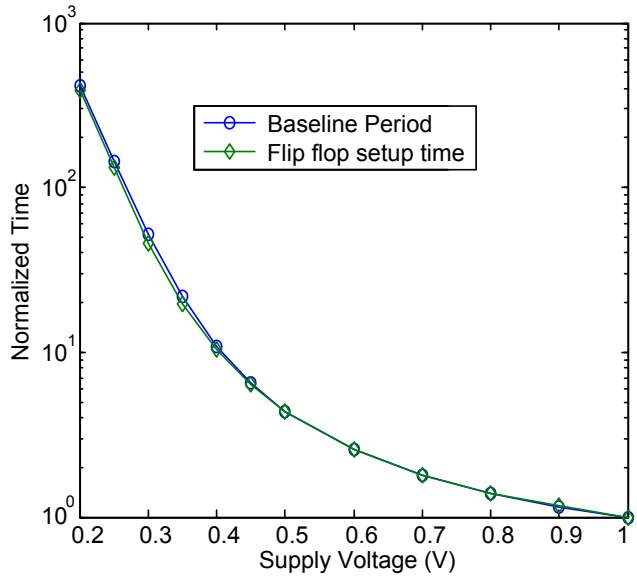


Figure 3.3: Baseline timing curve and normalized flip-flop setup requirement.

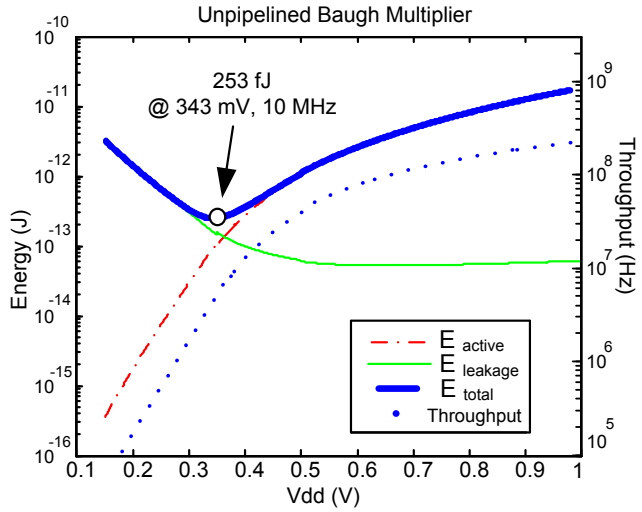


Figure 3.4: Comparing the estimated and measured timing and power of a multiplier.

The simulation framework can also be used with sequential circuits. Figure 3.3 shows the baseline timing curve along with the normalized setup requirement of a flip flop. To determine the setup time, the period between the data edge changing and the clock asserting was made larger and larger until the data latched. The critical delay of a sequential circuit is the sum of the combinatorial critical delay and the flip flop setup requirement. Because both of these normalized curves scale identically with a decreasing supply voltage, the framework can be used to estimate the critical delay of a sequential circuit. Hold times are not shown, because as the supply voltage decreases, the hold requirement decreases. The delay to the flip flop increases with reduced supply voltages; so as long as a circuit does not violate hold times at nominal voltage, it will not violate hold times at reduced voltages.

3.4.1 Large Circuit Test

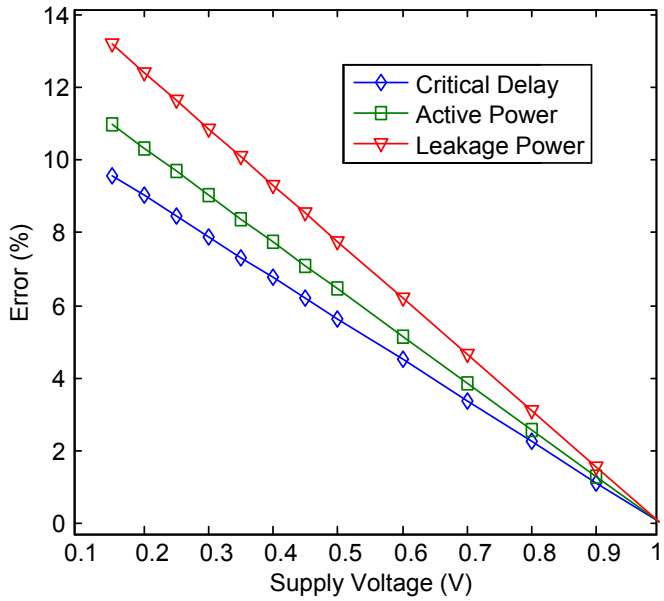


Figure 3.5: Error between the estimated and measured timing and power of multiplier.

All of the previous tests involved small circuits, so a large circuit was compared with the baseline as well. A 15,000-transistor multiplier was placed and routed and the parasitics were extracted and annotated. Pass transistor logic and gates with four or more inputs were excluded because of their poor reliability in the subthreshold region[28]. The back annotated multiplier was simulated

in SPICE from nominal voltage down to the minimum operating voltage. Figure 3.4 compares the measured delay and power of the multiplier with estimates obtained using the simulation framework. Figure 3.5 shows the error between the estimated and measured values. The error is curve fitted to make the linear trend more clear. The multiplier contains a number of large wires, gates with one to three inputs, and transistors with varying widths. Figure 3.5 shows that even with a large and varied circuit, the estimated result using a ring oscillator curve differs from the measured result by only 9 % for timing, 11 % for active power, and 13 % for leakage power at the lowest voltage. This error is small considering the framework only uses nominal values and at the lowest voltage, the multiplier is more than 1000 times slower. Another consideration is that it took more than a day to simulate the multiplier across the range of voltages in SPICE, while the simulation framework took only seconds to obtain an estimate.

3.5 Sample Applications

3.5.1 Architectural Comparisons

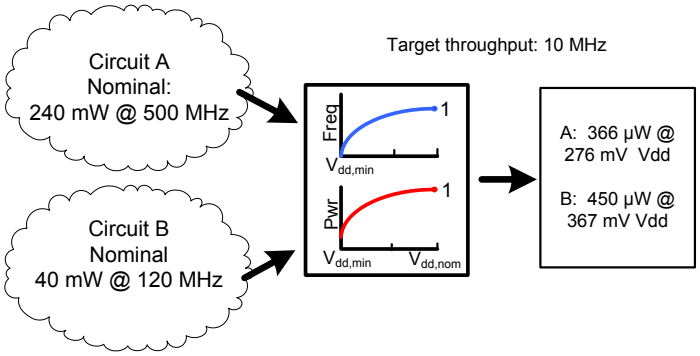


Figure 3.6: Using simulation framework to compare two architectures.

The simulation framework can be used to make quick architectural decisions without resorting to exhaustive simulations. Figure 3.6 shows two circuits, each with different power consumptions and frequencies. The target throughput of the application, 10 MHz, is a lot lower than what the circuits are capable of, so voltage scaling can be used to slow down the circuits and reduce the power consumption. The simulation framework can be used to determine which circuit would

consume less power when running at 10 MHz. Figure 3.6 shows that Circuit A would produce a 10 MHz throughput with a 276 mV supply and consume roughly two thirds the power of Circuit B. The simulation framework could produce this result in seconds, compared with simulating the two circuits exhaustively across a wide range of voltages.

3.5.2 Minimum Energy Voltage

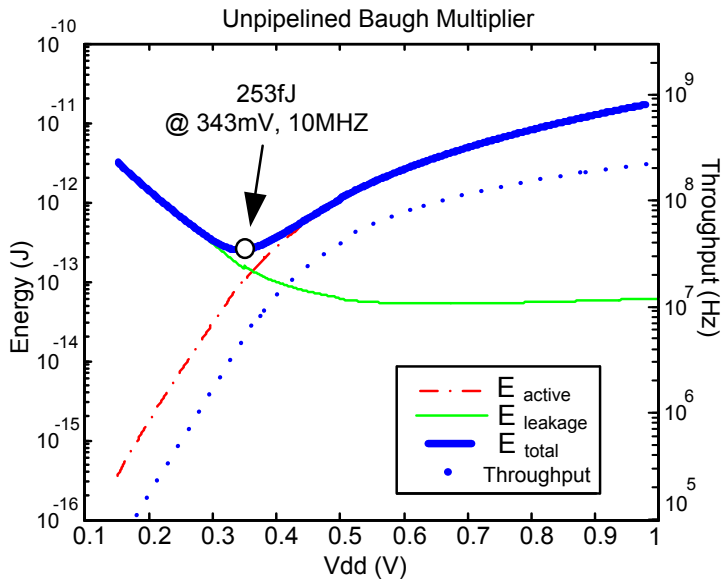


Figure 3.7: Minimum energy voltage of a 16 bit Baugh-Wooley multiplier.

Another application of the simulation framework is determining the minimum energy voltage of a circuit. Figure 3.7 shows the active energy, leakage energy, total energy, and throughput of a 16 bit multiplier across a range of supply voltages. When reducing the supply voltage and frequency of a circuit, the active energy per operation decreases. The leakage energy, on the other hand, increases due to the fact that the operation takes longer to complete, so the circuit spends more time leaking. There exists a point where the decrease in active energy due to voltage scaling is outweighed by the increase in leakage energy. From this point, any further decrease in supply voltage only leads to an increase in energy per operation. This voltage is known as the minimum energy voltage, and as shown in [2], it exists in the subthreshold region. Operating at the minimum energy voltage is useful for non-performance constrained applications that have little to no concern

about processor speed, and the simulation framework can be used to find it. With the use of the framework, only the nominal frequency, active power and leakage power of a circuit are needed to determine the operating voltage at which energy consumption is minimized. The framework gives curves for frequency, active power, and leakage power. The power curves can be divided by the frequency to determine energy, and the leakage and active energy can be added to form the total energy curve. The minimum of this curve is the voltage at which energy consumption per clock cycle or operation is minimized.

3.6 Conclusion

This chapter presented a framework to quickly characterize the frequency and power consumption of a circuit across a wide range of voltages, including the subthreshold region. One problem with simulating circuits with low supply voltages is that transistor level simulations can take a long time due to the increased simulation time, and accuracy requirements for the subthreshold region. The simulation framework addresses this issue by using only the nominal frequency and power of a circuit and multiplying them with normalized ring oscillator curves. The framework was tested against a number of designer controlled parameters including transistor widths, critical path lengths, gate inputs, and bus lengths, among others. In addition, the framework was used to estimate the power and timing of a 15,000-transistor multiplier. The estimate was accurate and only took a couple of seconds to obtain, while the SPICE simulations took many hours to complete. The speed and accuracy of this framework makes it useful for many applications including architectural comparisons, minimum energy point determination, and design space exploration.

4 Discrete Wavelet Transform using Subthreshold Voltage Technology

4.1 Introduction and Background

The Wavelet Transform (WT) is a method for analyzing the frequency content of a signal, and is similar to the Fourier Transform (FT) [9]. The wavelet transform, unlike Fourier transforms, can provide temporal information about when frequencies occur. It also provides better resolution for both high and low frequencies and has uses in many fields such as image processing. However, even in its more efficient forms, the wavelet transform requires a significant amount of processing power. To facilitate its use on battery powered system on chip applications or in signal processing, fast and efficient transform hardware is desirable. One portable application involving the use of the wavelet transform is devices that use the JPEG2000 codec. Cameras, digital picture frames, cell phones, and mobile internet browsers are all predicted to eventually use the new JPEG2000 codec instead of JPEG [7]. Portable sensing applications can also make use of the signal processing applications that use wavelet transforms.

In this chapter we propose a parallel implementation for the discretized form of the wavelet transform: the DWT. Because of its efficient parallel architecture with low overhead, this design is able to significantly benefit from voltage scaling to achieve energy efficiency. In our design, the number of parallel units is increased and the speed is reduced while maintaining a desired throughput. Our results show that the optimal operating voltage of the parallel units in terms of power consumption, is well below the threshold voltage, which is the voltage that turns the

transistors on. In this section, we provide the background to two of the topics related to this work: Discrete Wavelet Transform and subthreshold voltage operation.

4.1.1 DWT

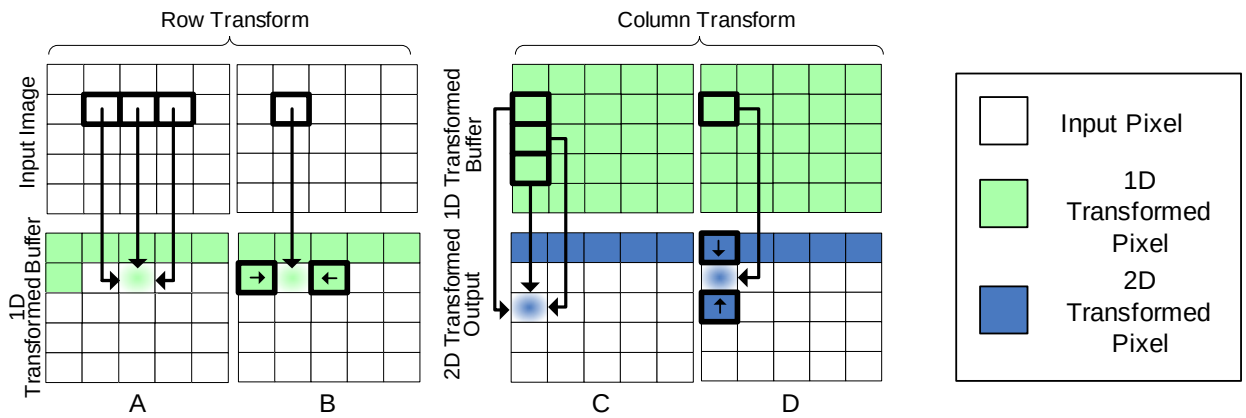


Figure 4.1: Row and Column DWT transform.

The Discrete Wavelet Transform (DWT) is a discretized form of the continuous wavelet transform, performed on 1-Dimensional (1D) and 2-Dimensional (2D) arrays. The original method used to perform a DWT is convolution, a method which requires a large amount of processing. The lifting scheme [10] was proposed as a method of reducing the amount of computation. Two lifting based techniques, the (5,3) and (9,7) transforms, were chosen as the transform coders for JPEG2000 due to their advantages in compression and flexibility over the methods used in JPEG. In this chapter, we focus on implementing the (5,3) transform that, unlike the (9,7) version, uses integer coefficients and therefore performs lossless transformation.

The 1D transform operates on a 1D array and produces an output that is the same size as the input. Equations (4.1) and (4.2) describe the 1D (5,3) DWT, and are derived using the lifting scheme in [10]. If a pixel's index is odd, a high pass transformation is performed on it. As seen in Equation (4.1), it depends on its two neighbors. The low pass calculation performed on even indexed pixels is dependent on the transformed high pass result of its two neighbors. This means that if a pixel is on an even index, it depends on its four closest *untransformed* neighbors. For data

at the edges, mirroring is used to produce a reversible result [21].

$$HP(2n+1) = Input(2n+1) - \frac{1}{2}[Input(2n) + Input(2n+2)] \tag{4.1}$$

$$LP(2n) = Input(2n) + \frac{1}{4}[HP(2n-1) + HP(2n+1) + 2] \tag{4.2}$$

With a 2D transformation, these equations will be applied twice to each pixel: once in the row direction and once in the column direction. A 2D transform operates on a 2D array by first treating all of the rows of an image as 1D arrays and performing a 1D DWT on them. The columns of the resulting transformed array are then treated with another 1D DWT. Figure 4.1 shows the progression of a 2D DWT. In the first step shown in parts A and B of the figure, the pixels are processed row-wise. In the second step shown in parts C and D of the figure, the columns are processed. The arrows in the figure represent data dependencies. In this figure, the original untransformed pixels are shown in white, while the row-transformed and row-and-column-transformed pixels are shown in green and blue respectively.

4.2 State-of-the-Art Pipelined Implementation of DWT

As can be seen from Equations (4.1) and (4.2), a 1D (5,3) transform requires 2 add/subtracts and a shift that is used for division by 2 or 4. The '+2' for the low pass step is a rounding operation and is present for JPEG2000 compatibility. It can be implemented using additional logic within an adder and therefore would not require another adder. The reference implementation, described in [23], breaks the DWT operations into three pipeline stages, where each stage represents a mathematical operation from Equations (4.1) and (4.2). The single processor scans and processes an entire row, then moves onto the next row and repeats.

The data is fed into the reference implementation serially from memory at a certain throughput. A single input pixel of a 2D-DWT is processed twice; once for the row transform and once for the column transform. When the processor is operating on an even row, it cannot perform a column

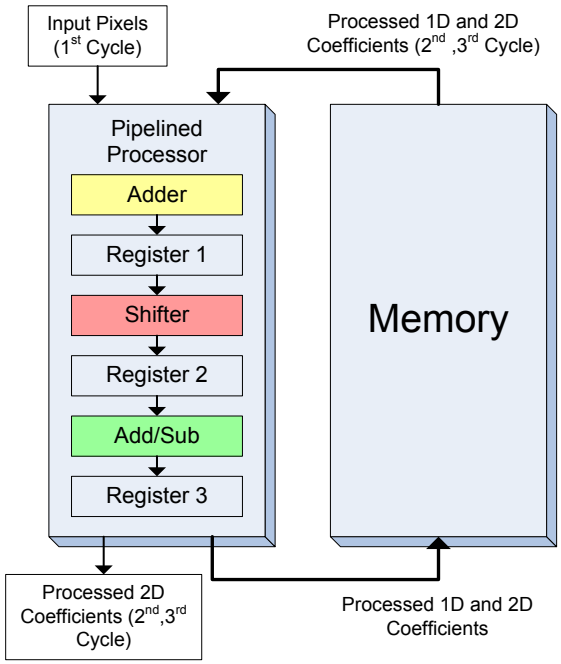


Figure 4.2: Reference implementation.

transform because the operation requires processed data from the next row, which is not available. The result of this dependency is that while the pipelined design is on odd rows, it must perform three transforms for every pixel: A row transform, a column transform for an odd row and an additional column transform for an even row. To facilitate the ability to perform a 1D DWT three times in one data clock cycle, the DWT processor runs at three times the speed of the data clock. Hardware utilization is not optimal because when the processor is working on even rows, 67% of the time no operations are being carried out.

4.2.1 Memory Organization of Pipeline Implementation

All column transformations require 1D row transformed data, and column transformations for even indices require column transformed data from the odd neighboring indices. This dependent data needs to be temporarily saved in an external memory bank. Three memory banks are required: one bank stores column transformed data for even column transformation, and two banks store row transformed data. These banks are implemented using SRAM. The pipelined design also requires 6 internal registers that save previous values and input values.

4.2.2 Potential Weaknesses

Using a single DWT processor to perform all of the row and column transforms has its drawbacks. There are three inputs into the DWT processor, but there are a number of different sources for these inputs. For row transforms, the inputs come from the input registers and internal registers. For column transforms, the inputs come from the memory banks. The edges and corners further complicate things because they are special cases that require mirrored data. The inputs are also different depending on if the processor is operating on an even or odd row or column. The end result is that each input to the DWT processor has up to 9 different possible sources. This requires a multiplexer for each input.

4.3 Proposed Parallel Implementation

4.3.1 Design Overview

Figure 4.3 shows a 1D parallel transformation system. Each column has its own dedicated (5,3) processing unit. This unit, which we call a DWT processor, performs the complete DWT operations mentioned in Equations (4.1) and (4.2) in one single clock cycle. The odd column (high pass) results depend on the untransformed data of their neighbors so the odd DWT processors are wired as such. The even columns (low pass) depend on the *transformed* data of their neighbors so they are wired to a register that is connected to the output of the high pass DWT processors next to them. This is relatively simple, but it only performs a 1D transformation. To perform a 2D transformation, the row transformed data needs to be saved and processed a second time. Our proposed parallel design uses another set of DWT processors for this purpose. We call the DWT processors that transform the input pixels *row processors*, and the ones that work on the result of these row processors *column processors*.

Figure 4.4 shows how row and column processors are organized in a simple 2x2 design. A chain of 5 registers is connected to the output of a row DWT processor. The row transformed data shifts

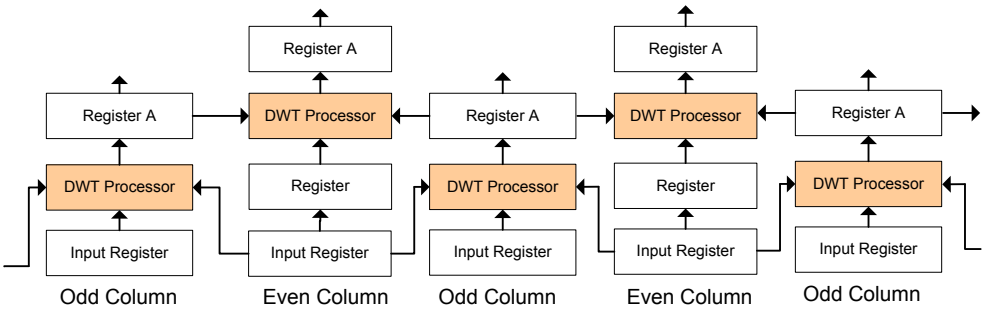


Figure 4.3: A parallel version of a 1D (5,3) processor.

up through these registers. A column DWT processor connects to these and performs a second DWT on the row transformed data. Like the row transform, a column high pass and column low pass result is produced. During one clock cycle, the column processor is connected to the bottom three registers of the register chain. These registers all contain row transformed data and the result is a column high pass result. Because the column low pass transformation requires the already processed column high pass result, when a column high pass result is produced, it is inserted into the register chain. During the next clock cycle, the column processor is connected to the top three registers. These three registers contain row transformed data in the middle, and column high pass data in the top and bottom. The result of this column transform is a low pass result. Thus, in two clock cycles a 2D high pass and low pass result is produced; or in other words, one 2D transformed pixel is produced in each clock cycle. Note that the even and odd columns share a register and odd and even columns are wired slightly differently.

The parallel design does not require an external memory module or SRAM. The only memory units used are flip flop based registers within the parallel units. The control of these registers is very simple and usually involves shifting a value from a previous register, and shifting out a value to the next register. There is no need for the address decoding and complicated wiring that SRAM based modules require.

4.3.2 Working with Large Images (Striping)

We refer to the maximum image width a design can transform without breaking up the image as the *width of the design*. For the pipelined design, the width is determined by the width of the memory

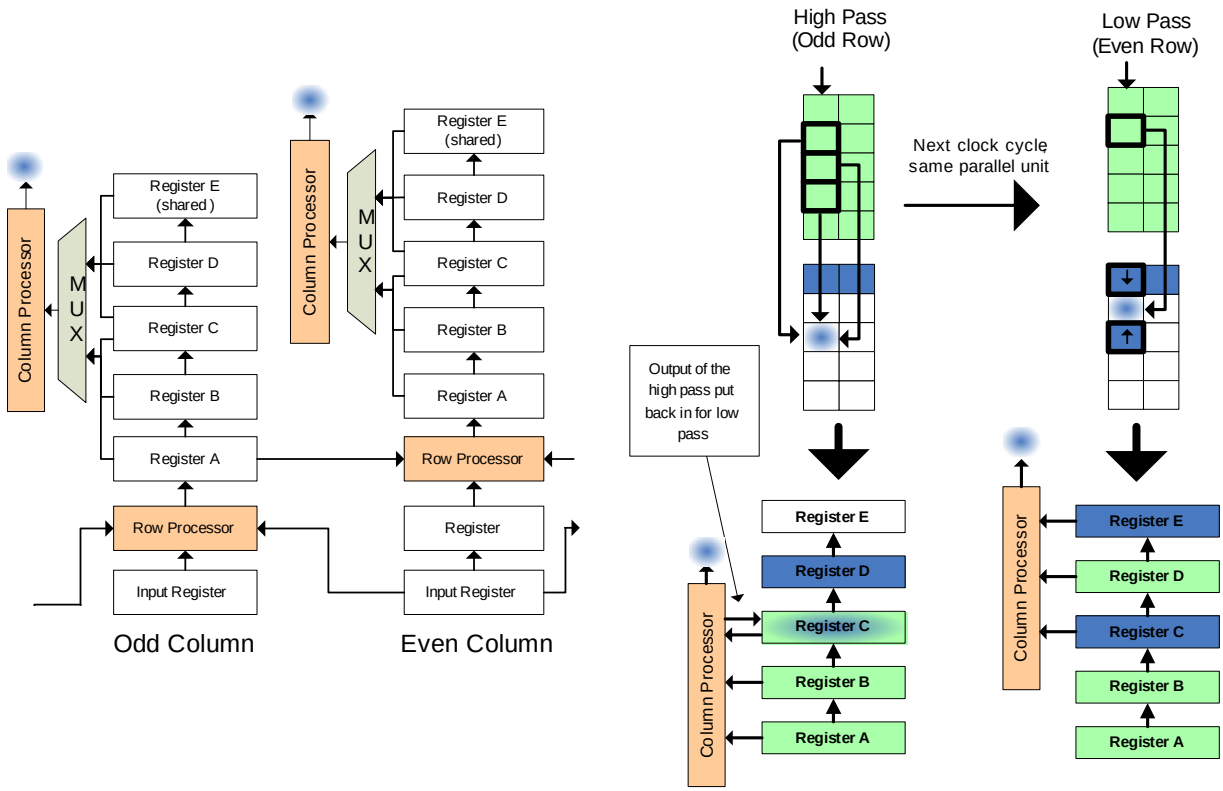


Figure 4.4: 2D DWT Parallel System

banks used to store intermediate values. For the parallel design, the width is the number of parallel units. When dealing with an input image that is n pixels wide, it is possible to transform the image with an implementation that has a width less than n. This can be done by striping the image. As shown in Figure 4.5, striping breaks the image into stripes and processes all of the columns of a stripe in the image before moving onto the next stripe.

There is a penalty associated with striping: a row transform operation on the edge of a stripe requires data from the stripe next to it. If the edge of the stripe is an odd column, it requires the pixel next to it on the neighboring stripe. If the edge is an even column, it requires two pixels from the neighboring stripe. Assuming that the width of the processor is a power of two, there is always one edge that is on an even column, and one edge that is on an odd column. These additional dependencies to the left and right of the stripe must be loaded from memory to the internal storage of the design.

The penalty associated with striping is an increase in bus power consumption. Additional parallel units are not needed when striping because the extra data is only needed to satisfy a dependency

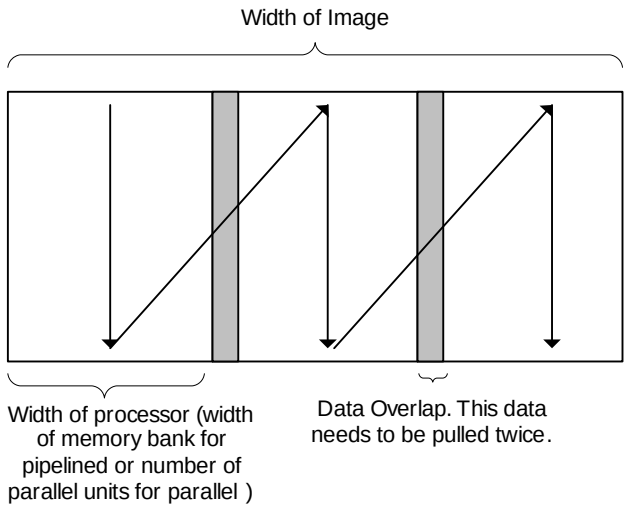


Figure 4.5: Striping an input image

of the calculations of edge pixels. For even numbered columns on the edge of a stripe, *row transformed* data is required as a dependency, so two pixels from the neighboring stripe are needed and that data needs to be row transformed. This requires an additional DWT processor. As mentioned earlier three extra pixels per row are needed, so three additional registers are needed on the bus. The bus will also have to run faster so it can keep up with filling the additional registers. Equation (4.3) lists the number of extra pixels per row required to stripe, given a width of the image and a width of the processor. Equation (4.3) lists the new clock speed the bus must run at to supply the additional registers.

$$ExtraPixelsPerRow = \lfloor \log_2 (ImageWidth) - \log_2 (ProcessorWidth) - 1 \rfloor$$

$$NewBusDataRate = \frac{(ProcessorWidth + ExtraPixelsPerRow) * OldBusDataRate}{ProcessorWidth} \tag{4.3}$$

The striping penalty would then be the increased power consumption as a result of running the bus faster, adding three registers to the bus, and adding an additional DWT processor.

4.3.3 Data Distribution to the Parallel Units

The assumption for input data of the parallel design is the same as the reference pipelined design: image data will be delivered serially at a given data rate. Some sort of data delivery system is required that distributes the serial data to all of the parallel units. The simplest type of data distribution system is the bus shown in Figure 4.6A. Each input register of a parallel unit is connected to the bus and the serial data is sent over the serial bus at the overall data rate. A global column counter is also sent to every parallel unit. When the global column counter equals the column number of a parallel unit, it latches in the data from the bus. As mentioned in Section 4.4.3, the V_{dd} of the parallel units is reduced in order to reduce the power consumption. Moreover, it will be shown in the results section that the optimum operating voltage of these parallel units is well into the subthreshold voltage region. However, it is not possible to slow down the input registers of the parallel units as much as the DWT processors because the inputs to these registers change at a fast data clock rate and using slow registers will result in setup and hold errors. On the other hand, fast input registers consume a considerable amount of power, which squanders the gains from power reduction of DWT processors.

In order to solve this problem, we split the bus into two or more slower buses. This is the reverse of multiplexing and is done using a demux as shown in Figure 4.6. If the original data bus is split into two busses, for example, the speed of each of the new two busses is half of the original bus while the two busses collectively carry the data that is delivered through the original bus. The benefit of splitting the bus is that the input registers to the parallel units have more time to latch the data and can be slowed down in order to reduce the power consumption. It is noteworthy that when the demux selects one side, the other side goes to an indeterminate state, so a fast register is used to latch the value onto the sub bus. The area and power consumption of this data distribution module is the overhead associated with increasing the parallelism in our design and it increases as more parallel units are added. In the Results section, we provide the result of an analysis on the optimum number of bus splits.

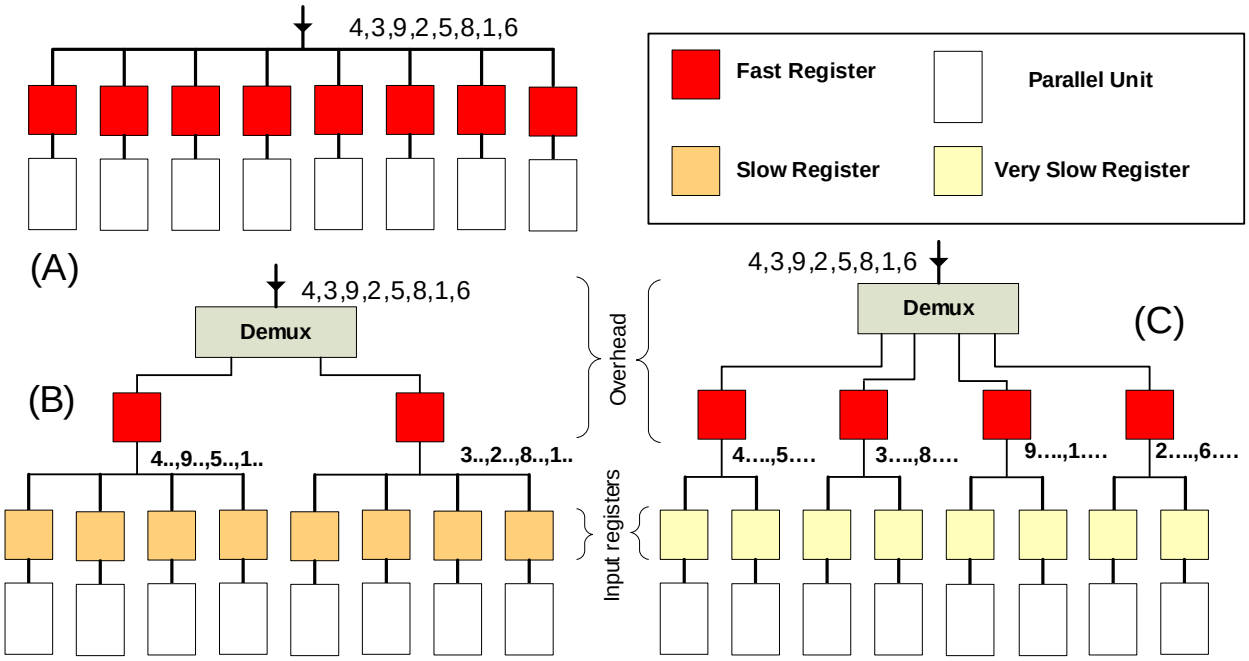


Figure 4.6: Bus Splitting: A shows a simple bus. B shows a single split bus. C shows a bus split four times.

4.4 Methodology

4.4.1 HDL Tool Chain

Table 4.1 gives the HDL tool chain used to perform the experiments.

Table 4.1: HDL Tool Chain

Design Aspect	Tool
HDL	Verilog
Simulator	Synopsys VCS
Standard Cell Library	OSU 0.18 μm [22]
Synthesis	Synopsys Design Compiler
Power Measurements	Synopsys PrimeTime PX

4.4.2 SRAM Simulation

The pipelined design contains three memory banks. Modern memory banks are usually implemented as SRAM modules. The OSU standard cell library does not have an SRAM model and synthesizes memory structures inefficiently with standard flip-flops and primitive gates. To model

these memory elements more realistically CACTI 4.2 was used to obtain power, area, and speed [26]. CACTI is an SRAM cache modeling tool that is used extensively in computer architecture research. To model the structure of a memory bank, the CACTI tool was setup to model a cache that is one-way set associative and has zero tag bits. These settings eliminate the tag portion of the cache. The CACTI tool only allows a minimum entry length of 8 bytes. Word length has a significant effect on power and area of memory. Therefore, curve fitting was applied to the CACTI data of memory structures with varying word lengths and the area and power of the memory unit was extrapolated.

4.4.3 Achieving Maximum Energy Efficiency

There are $2n$ DWT processors or n pairs of row and column processors (called DWT pairs) in our proposed parallel design where n is the width of the design (defined in the previous section). As stated before, after the initial warmup time, each DWT pair transforms two pixels in each cycle. Therefore, the throughput of the parallel design can be calculated as shown in Equation (4.4):

$$ParallelThroughput = n * SpeedOfDWTProcessor \tag{4.4}$$

Since the throughput depends on both the width of the design and the speed of DWT processors, it is possible to change their values and still maintain the same throughput. The goal of our design is to maximize energy efficiency for a given throughput. In order to achieve this, we compute the power consumption of the parallel design while varying n and maintaining the same throughput[1]. As we increase n, the speed of the DWT processors is reduced. We use voltage scaling techniques in order to achieve this. Figure 4.7 shows the algorithm used to find the most energy efficient design.

[1]The DWT can be performed on an image of any width, but one of the driving applications of the wavelet transform is JPEG2000. In the JPEG2000 compression process, the output of the wavelet transform goes into an encoder that uses Embedded Block Coding with Optimal Truncation (EBCOT). The maximum size of a code block in JPEG2000 EBCOT is 256, so the image has to be split into blocks that are at most 256x256. Because of this requirement, for the purposes of this design, the maximum size of an input image will be 256.

1. For *n* = 2, 4, 8, 16, 32, 64, 128, 256
   DWT_speed = *Throughput* / *n*
   a) Use voltage scaling trends to determine the operating voltage, *V*, of DWT processor to make it run at *DWT_speed*
   b) Compute the power consumption of the DWT processor at *V* as *DWT_power*
   c) Record the power consumption of the system from *DWT_power* x *n*
2. Identify *n* that, after including overhead, results in minimum power consumption of the system.

Figure 4.7: Finding the most energy efficient parallel design.

4.4.4 Supply Voltage Scaling and Splitting the Bus

For this experiment, the power consumption of the bus after splitting will be determined for a number of bus widths. The width of the bus is defined as the number of input registers attached to it. First, the power consumption of the simple bus at nominal voltage running at its maximum speed will be determined. The supply voltage (V_{dd}) of the bus will then be scaled down until the bus's maximum frequency is the target throughput. This V_{dd} will be referred to as the *Throughput* V_{dd}. The bus will be split according to the procedure shown in Figure 4.6. The fast registers and demultiplexers in Figure 4.6 cannot be V_{dd} scaled any further so the supply voltage to it must remain at the throughput V_{dd}. The V_{dd} of the input registers can be scaled down further because the effective data rate on the sub bus they are attached to is one half of the original data rate. The input registers are V_{dd} scaled until their maximum frequency is equal to the effective data rate on the sub bus. The bus is then split again and the experiment is repeated. The bus is split until the number of splits equals the width of the bus. This experiment is performed for bus widths of 8, 16, 32, 64, 128, and 256.

4.5 Results

4.5.1 Synthesis Results

Table 4.2 presents the basic synthesis results for the reference pipelined implementation excluding the SRAM module as well as a pair of parallel units excluding the data distribution module. The

delay and power consumption reported are from operation at nominal voltage.

	Pipelined Design	Parallel Design
Notes	Not Incl. SRAM	2 parallel units
Critical Delay	1.67 ns	2.97 ns
Power	39.3 mW @ 200 MHz	71.8 mW @ 336 MHz
Area	0.00426 mm^2	0.00716 mm^2

Table 4.2: Pipelined and Parallel Results at Nominal Voltage (1.8 V).

4.5.2 Analysis of Voltage Scaling the Parallel Units

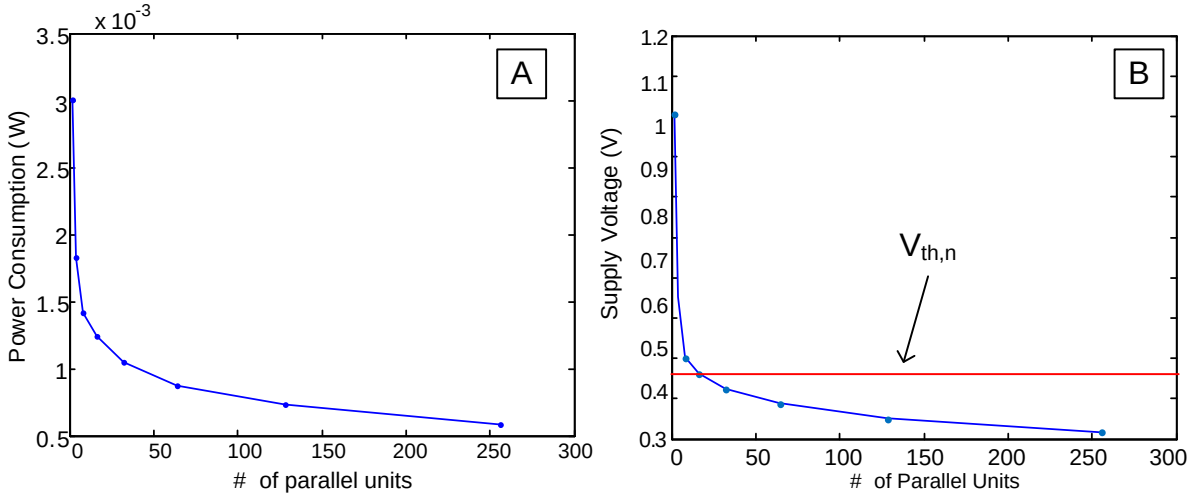


Figure 4.8: A. Power consumption for a given number of parallel units @ 200 MHz, no overhead. B. V_{dd} of the parallel units.

Figure 4.8.A shows the power consumption of the parallel DWT units, for a given number of parallel units. The throughput is constant at 200 MHz, and the graph does not include the data distribution module. Figure 4.8.B shows the V_{dd} that each parallel unit is running at for a given number of parallel units. The power consumption shows diminishing returns as more parallel units are added. However, for these graphs, increasing the number of parallel units is always beneficial as we are not yet including the overhead associated with the data distribution module.

4.5.3 Analysis of Voltage Scaling the Data Distribution Module

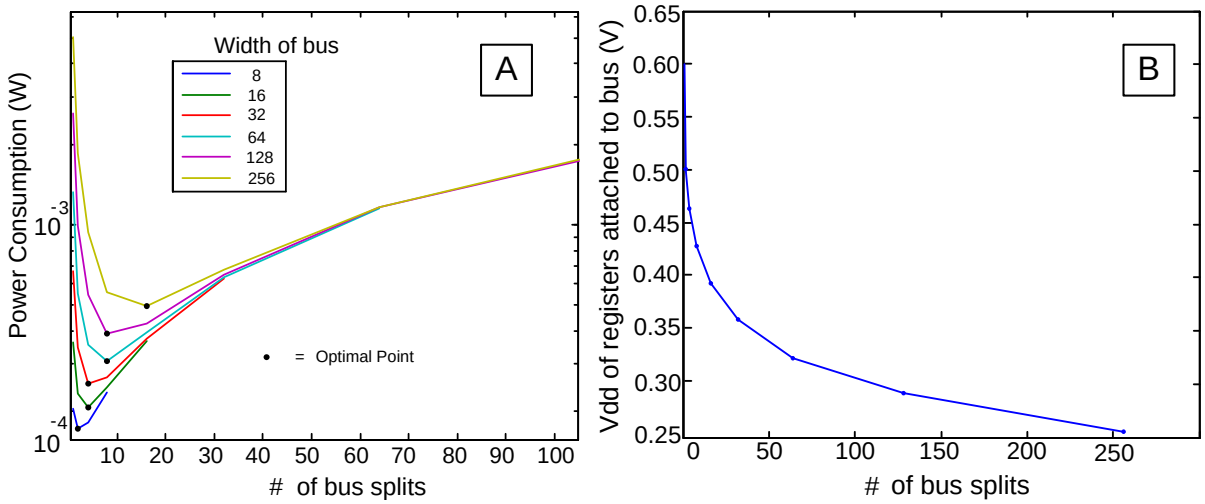


Figure 4.9: A. Number of bus splits vs. power consumption. B. Optimal V_{dd} of the registers attached to the buses.

Figure 4.9.A shows the number of splits on the bus vs. the power consumption, for a variety of bus widths. It shows that there is an optimal power consumption for a given number of splits on the bus. Each time a bus is split, a lower V_{dd} can be used to power the registers. Figure 4.9.B shows the V_{dd} of the different buses, for a given number of bus splits. Running a number of different power networks across a design would end up being prohibitive because of routing issues and the need for level converters. For this design, we will find a bus that can run at the same V_{dd} as the parallel units. This will be referred to as the V_{dd} Matched Split Bus. Because of this V_{dd} restriction, there are only two options: the simple bus, which runs at nominal voltage, and the V_{dd} Matched Split bus. Figure 4.10.A shows the power consumption of both a simple bus and the V_{dd} matched bus for parallel unit counts from 1 to 256. This graph shows that the V_{dd} Matched Split Bus is always superior no matter the number of parallel units. Because of this, the V_{dd} Matched Split Bus will be used for all total power consumption calculations.

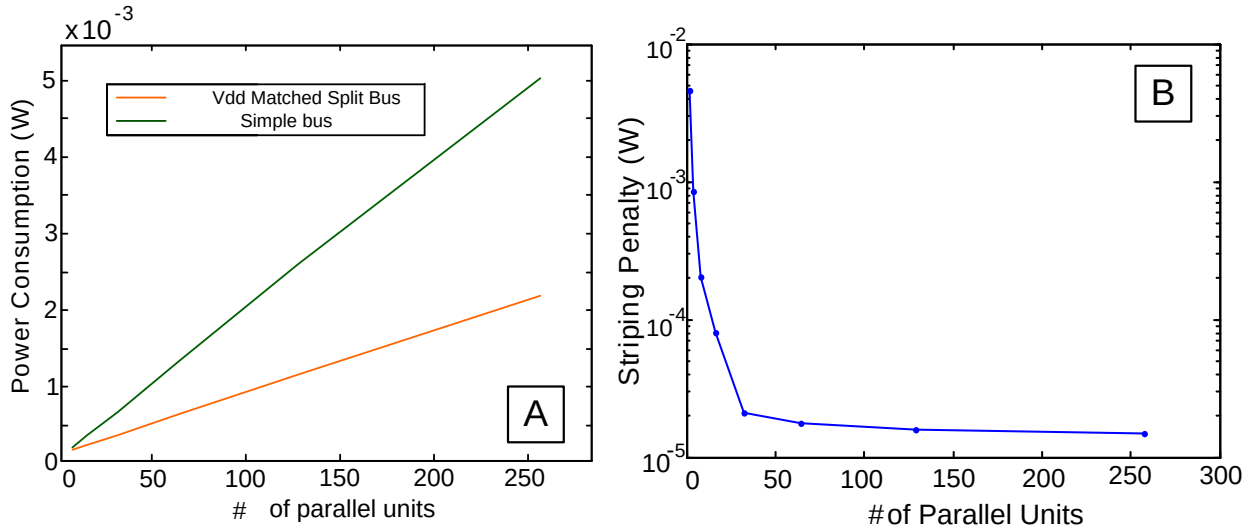


Figure 4.10: A. Simple bus vs. V_{DD} matched split bus B. Striping penalty for a given number of parallel units.

4.5.4 Striping Penalty

Figure 4.10.B shows the result of the striping penalty in terms of power consumption. It can be seen that as the number of parallel units or width of the design increases, less striping is needed and therefore, less power is wasted for this purpose.

4.5.5 Total Power

Two different total power consumption numbers are calculated in Figure 4.11. The first power consumption includes the striping penalty, and the second one does not. Equation (4.5) shows how the total power consumption is calculated when striping is included.

$$TotalPower = ProcessorPower + VddMatchedSplitBusPower + StripingPenalty \tag{4.5}$$

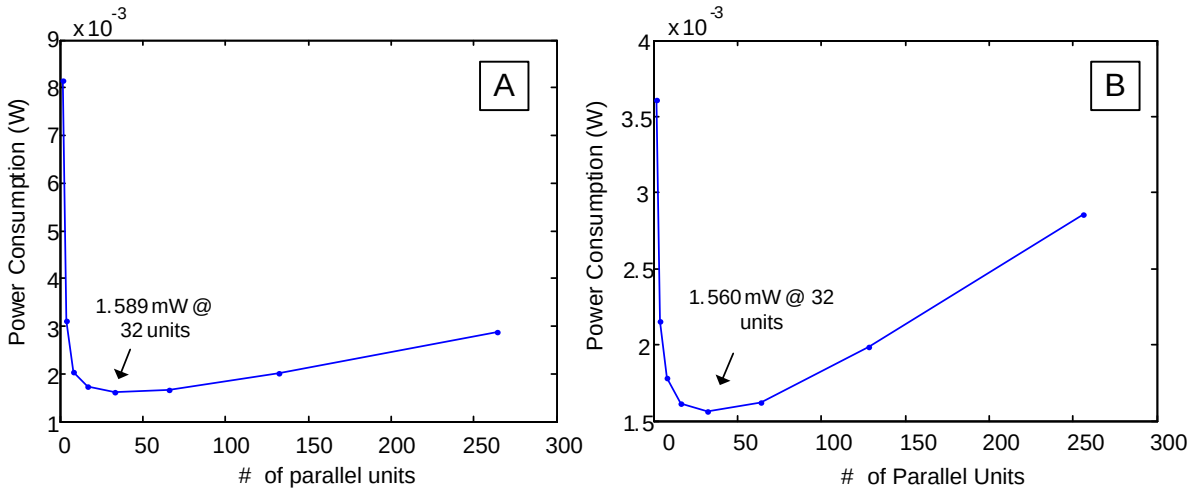


Figure 4.11: A. Total power consumption of parallel architecture without striping. B. Total power consumption of parallel architecture with striping.

4.5.6 Comparisons

Table 4.3 shows the results of the optimal parallel design, with the striping penalty included, compared to a pipelined design that stripes and one that does not. The V_{dd} scaled parallel design gives a 26.3x reduction in power consumption compared to the reference pipelined implementation. If the pipelined design is not striping and uses a 256 word memory bank, then the optimal parallel design takes up roughly half the area of the pipelined design. If the pipelined design is striping with 32 pixel wide stripes, then the area of the optimal parallel design is roughly 4 times that of the pipelined design.

Table 4.3: Optimal Parallel Design and the Pipelined Design, Throughput = 200 MHz

	Parallel Design	Pipelined No Striping	Pipelined Striping
# of Units	32	1	1
Processor Power	1.59 mW	39.3 mW	39.3 mW
SRAM Power	n/a	9.80 mW	1.72 mW
Total Power	1.59 mW	49.1 mW	41.0 mW
Processor Area	0.121 mm^2	0.0426 mm^2	0.0426 mm^2
Bus Area	0.0630 mm^2	n/a	n/a
SRAM Area	n/a	2.129 mm^2	0.258 mm^2
Total Area	1.208 mm^2	2.172 mm^2	0.300 mm^2

4.6 Conclusion

In this chapter, we presented a new parallel architecture for a 2D lifting-based discrete wavelet transform processor. By increasing the number of parallel units and implementing voltage scaling, this design consumed 1.59 mW compared to the 39.3 mW of a single state of the art pipelined DWT processor, with both running at the same throughput. This is a 26x reduction in power and the operating voltage of our design was in the subthreshold region of 386 mV. The data delivery to the parallel system was also explored. It was found that by adding registers and a demultiplexer to the data delivery system, the supply voltage to certain portions of the bus could be run at the V_{dd} of the parallel units, and the overall power consumption could be reduced further. Future work would include doing a complete ASIC design and taping out a chip. This would provide a more accurate real world model of how a scaled down supply voltage affects the power consumption.

5 Hybrid Super/Subthreshold Fast Fourier Transform

5.1 Introduction

As Charles Van Loan wrote in his book ”The Fast Fourier Transform (FFT) is one of the truly great computational developments of this century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very difficult without the FFT” [16]. The FFT has had a widespread application in traditional fields such as communication and manufacturing. The advent of wireless sensor networks has created even more applications for the transform. Sensor nodes are employed to monitor the environment and report interesting data or significant events [1]. The FFT can be used to analyze the raw data in order to identify such events. This is especially important for situations where large amounts of data are collected, but there is only an occasional need to report back data. Since the sensor nodes run on either a battery or a limited amount of scavenged energy, and communication costs are still the dominant factor in power consumption, it is usually advantageous to process the data locally and transmit only a message if an interesting event is detected. In an example scenario, a node with an acoustic sensor, which is employed in a field to detect passing vehicles, collects sound samples periodically. It then analyzes the collected data using an FFT to determine if its frequency content includes components representing frequencies found in a moving vehicle such as a humming engine frequency. If so, it records a significant event and transmits the data to a central station or other nodes depending on how the sensor network is implemented.

It is also highly desirable to be able to scale the throughput of an FFT operation for a sensor node

[12]. Increasing the amount of transformed data yields more resolution in the frequency domain. However, high quality FFTs are computationally intensive and consume high levels of power, which is impractical for wireless sensor nodes. It is therefore suggested that during idle periods, a low throughput FFT is used that consumes less power. When a significant event is suspected, the throughput of the FFT is ramped up so that the data can be analyzed more diligently.

In this chapter, we present a low-power parallel implementation of the FFT with scalable throughput. Our novel FFT design partitions a traditional butterfly FFT architecture into two regions, namely memory banks and processing elements, so that the operating voltage of the former region is above the threshold voltage while that of the latter region is in the subthreshold region. Our proposed design is able to deliver the same throughput as a traditional design while consuming 68 % less power. Furthermore, we study the effectiveness of this method in a variable throughput application. We compare our method with other methods used for scaling the throughput, namely voltage scaling and clock scaling. Our results indicate that in an example scenario, if all these designs are running on 2 alkaline AA batteries and spend 15 % of their time in high quality mode and the rest in the low quality mode, our design can last up to 111 days while the other two last only 59 and 40 days respectively. The cost of the decrease in power is a 5x increase in area.

The rest of this chapter is organized as follows: Section 5.2 includes the fundamentals of the FFT operation. In Section 5.3, we present our novel parallel FFT architecture based on the traditional butterfly design. Section 5.4 describes our employed methodology to carry out the experiments while Section 5.5 presents the results of these experiments. Finally, in Section 5.6 we present the conclusions and future directions of this study.

5.2 Background

5.2.1 Fast Fourier Transform

The Fast Fourier Transform (FFT), formulated by Cooley and Tukey [8], is an efficient method for calculating the frequency content of a signal. The number of samples in the signal, N, determines

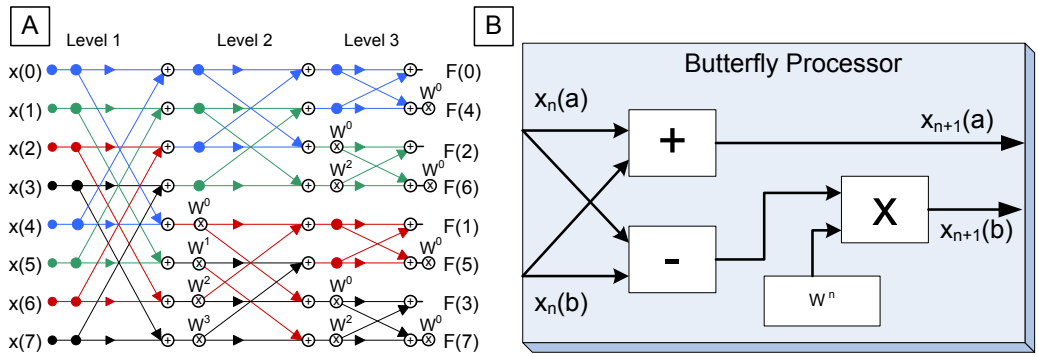


Figure 5.1: A. Signal Flow Graph of 8 pt. FFT B. Butterfly processor

the frequency resolution and quality of the Fast Fourier Transform. An increase in the number of data points yields more frequency resolution, but takes more computation as the complexity of this transform is equal to $O(N * log_2(N))$. Equation 5.1 presents the formulas that define the FFT.

$$X_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2m} W^{(2m)k} + \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} W^{(2m+1)k} \tag{5.1}$$

$$W^n = e^{-\frac{2\pi i}{N} n} \tag{5.2}$$

Equation 5.1 breaks up an N point FFT into the sum of two $\frac{N}{2}$ point FFTs. These $\frac{N}{2}$ point FFTs can then be broken up again and again leading to a fast recursive implementation of a DFT. The W coefficients are constants equal to nth roots of unity in the complex plane, traditionally called twiddle factors. Figure 5.1A shows the signal flow graph of an 8-point FFT. There are 3 levels in an 8-point FFT, corresponding to the $log_2(N)$ term in the complexity. There is a repetitive pattern in the FFT signal flow graph that looks like a butterfly. Each butterfly contains a complex addition, subtraction, and multiplication. It can be seen from the figure that there are 4 butterfly computations in each level, which is half of 8 data points and corresponds to the N term in the complexity. Overall there are $(N * log_2(N))/2$ butterfly operations in this FFT implementation[1]. It is highly impractical to implement this signal flow graph in hardware for a 1024-point FFT due to its large area requirement, so hardware must be reused to perform these calculations. One possibility is to

[1]There are other ways to implement FFT. However, the Cooley-Tukey algorithm discussed in this chapter is by far the most common implementation of FFT due to its efficiency.

use a single butterfly element and compute all the $(N * log_2(N))/2$ butterfly operations serially. Intermediate values can be stored in and recalled from a memory bank.

5.3 Implementation

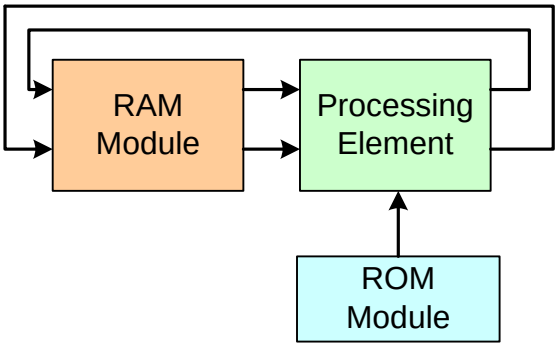


Figure 5.2: FFT Architecture

The 1024 point FFT architecture used in this chapter is based on [19] and is implemented in a 90 nm technology. It consists of three major modules: RAM, processing element, and ROM. The processing element (PE) performs the FFT calculations and contains at least one butterfly processor shown in Figure 5.1B. The butterfly processor implements a complex addition, subtraction, and multiplication. The multiplication is implemented using a Booth multiplier and the addition is implemented using efficient carry select adders. The architecture processes 32 bit complex numbers, where 16 bits represent the real portion and 16 bits represent the complex portion, all in the Q15 fixed point format.

The ROM module stores the constant twiddle coefficients while the RAM module is responsible for storing inputs, outputs, and intermediate values. The PE takes in three inputs: the first two inputs are either the input signals, or the intermediate values that are the result of a previous butterfly operation. The third operand is a constant W^n coefficient, taken from the ROM. The W^n coefficients are arranged around the unit circle in the complex plane, but the FFT only uses coefficients with positive imaginary parts. There is also symmetry of the coefficients on the left half and right half of the unit circle, so overall 256 32-bit coefficients are needed in the ROM

for a 1024 point transform. The architecture implements the FFT using decimation in frequency, which means initially, the RAM holds the 1024 32-bit input values, and they get replaced with intermediate and eventually output values as the FFT progresses.

5.3.1 Parallelization and Throughput Scalability

It is well-known that the parallelism present in applications such as the FFT can be utilized to improve the energy efficiency of the system without sacrificing performance [6]. In our design, we exploit the unique characteristics of subthreshold operation in this context. Traditional parallel designs focus on increasing the throughput of a system. Our design exploits parallelism in a different way, by adding parallel units and slowing them down to match the original throughput. To slow down the parallel units, the supply voltage is reduced which greatly reduces the power consumption. We will show that the optimal supply voltage for the parallel FFT architecture is in the subthreshold region.

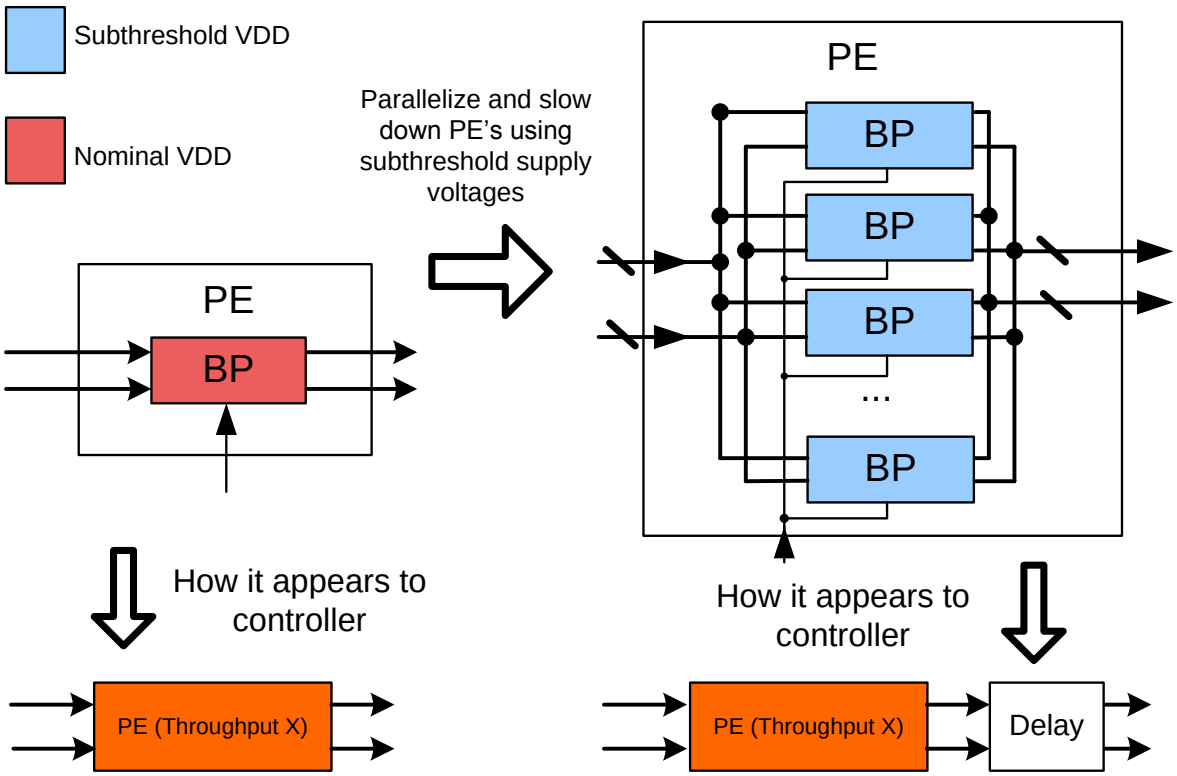


Figure 5.3: Methodology for parallelization of the PEs.

Our goal is to design an FFT architecture to operate with minimum energy given a desired level of performance. In order to achieve this, the number of butterfly processors (BPs) within the processing element (PE) is increased and the supply voltage to the processors is decreased such that the throughput remains constant, as shown in Figure 5.3. Even though a processing element may contain a number of butterfly processors, it must still only have three inputs and two outputs. The processing element, therefore, needs a data distribution bus that distributes the serial data from the RAM to each butterfly processor, and a collection bus that forms the processed data back into a serial stream. The PE uses a demultiplexer and a multiplexer to accomplish the data distribution and collection respectively. Data is distributed in a staggered fashion to the BPs, and is collected on the other end in a staggered fashion. The supply voltage of the BPs is scaled such that the existing data in a BP is processed before new data is sent to it. The PE, therefore, exhibits the same throughput no matter how many BPs it contains. The only difference is that there is increased latency in the PE, which grows linearly with respect to the number of BPs.

The parallelization lends itself well to a scalable throughput design. At maximum throughput, all of the BPs in the PE are activated and are processing data. To reduce the throughput, only some of the BPs are active and the rest are powered off. Ideally, the operating voltage of these remaining BP's should be adjusted to achieve maximum energy efficiency. However, as we will show in Section 5.5, the benefits of this ideal method does not justify the extra burden of adding voltage scaling capability to the circuitry. Instead, we simply power off a certain number of BP's while leaving the rest to run at the same speed. We call this method *active unit scaling*.

With more than one BP running in parallel, one can envision a design where the RAM bank is split into multiple banks, each running at a lower speed than the original RAM and feeding the slow-running BP units. As an example, the 1024-word RAM bank can be split into two separate 512-word banks. Both of these RAM modules, i.e. the single bank or dual bank, are capable of supplying data at the same rate, but each 512-word RAM banks is now required to run at half the speed of the original one. Because of the relaxed speed requirement, the supply voltage of the two 512-word banks can be reduced, resulting in reduction of the RAM power. The paid penalty is the

area taken up by the additional controller logic in the second bank. We study the practicality of this method and present our findings in the results section.

5.4 Methodology

Table 5.1 shows the tool chain used for synthesis and simulation of our design. In the rest of this section, we present the detailed methodology for the two major experiments done to study our proposed design.

Table 5.1: Tool Chain

HDL Language	Verilog
HDL Simulator	Synopsys VCS
Standard Cell Library	Industry level 90 nm
RAM/ROM memory compiler	Industry Level RAM/ROM Compiler
Voltage Scaling Characterization	Synopsys HSPICE, NanoSim
Power Simulation	Synopsys Primetime PX

5.4.1 Minimum Energy Architecture Experiment

The purpose of this experiment is to compute the optimum number of parallel butterfly processors as well as their operating voltage in order to achieve minimum energy consumption for a given throughput. Since the throughput is kept constant, the energy consumption is directly related to the power and we use these two terms interchangeably. To carry out this experiment, we use the trends presented in Section 3.2, which give the power consumption and the frequency (or throughput) of a BP for a given supply voltage. The inverse of this is used to determine the supply voltage — and then, power consumption — of a BP running at a given throughput. The throughput of the processing element with a single butterfly processor at nominal voltage is referred to as $TP_{Nominal}$, and is the target throughput for the minimum energy architecture. It is possible to increase the number of BPs in the PE and maintain $TP_{Nominal}$ by reducing the supply voltage of the BPs. Herein, n will refer to the number of parallel BPs in the PE. In order to achieve $TP_{Nominal}$ in a processing element, the throughput of each individual butterfly processor must be $\frac{TP_{Nominal}}{n}$. Using the characterization curves, the supply voltage that yields a BP throughput of

$\frac{TP_{Nominal}}{n}$ will be determined and the power consumption at that speed will be noted. The total power consumption of the PE is the power consumption of a BP at $\frac{TP_{Nominal}}{n}$ throughput times n. The power consumption of the bus needed to distribute and collect data from n BPs will also be determined and added to the BPs' power consumption. The *minimum energy architecture* will be defined as the FFT architecture with n BPs that consumes the least amount of power, while maintaining a throughput of $TP_{Nominal}$.

5.4.2 Throughput Scaling Experiment

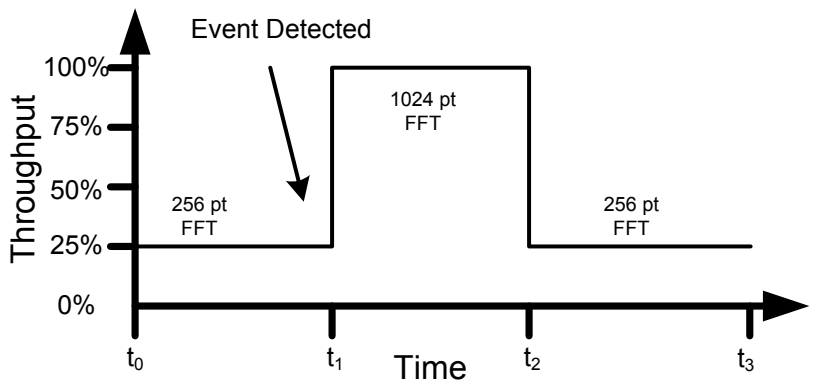


Figure 5.4: Example sensing scenario.

As stated in the introduction, there is a desire to dynamically scale the throughput of the FFT architecture. During idle times, fewer data points can be used in the FFT, and the throughput of the FFT processor can be reduced. Figure 5.4 shows an example scenario in a wireless sensor network. From $t0$ to $t1$, the hardware is performing a 256 point FFT. The throughput of the hardware is one quarter of the throughput at max speed. At $t1$, an event is detected and the hardware shifts to a 1024 point FFT. At this point, the hardware is running at its maximum throughput. From $t2$ to $t3$, the hardware goes back to a 256 point FFT. The duty cycle of the FFT hardware will be defined by Equation (5.3), or the time the hardware spends at max throughput divided by total time.

$$DutyCycle = \frac{t_2 - t_1}{t_3 - t_0} \tag{5.3}$$

There are three different methods that can be used to reduce the throughput of the FFT hardware and save power. The first method is clock scaling, where the clock speed to the PE is reduced. The supply voltage remains at nominal 1V. This will be done to the $n = 1$ FFT architecture (one BP per PE). The second method is dynamic voltage scaling, where the clock speed*and* supply voltage to the PE are reduced. This will also be done to the $n = 1$ FFT architecture. The third method will be the active unit scaling mentioned in Section 5.3.1. The minimum energy architecture will be used for this method. The ROM and RAM are kept running at nominal voltage for this experiment. The three methods of voltage scaling will be tested over various throughputs and the power consumption will be determined. It will also be determined, for a given duty cycle defined by Equation (5.3), how long each of the throughput scaling methods can last on two alkaline AA batteries (at 1500 mAh each) while executing the sensing scenario in Figure 5.4.

5.5 Results

5.5.1 Synthesis Results

Table 5.2 shows the results of the memory compiler and the synthesized Verilog code at nominal voltage (1V) and maximum speed.

Table 5.2: Synthesis Results of FFT Architecture

Butterfly critical path	4.43 ns
Butterfly throughput	222 MHz
Butterfly power consumption	19.6 mW @ 222 MHz
Butterfly area	0.0498 mm^2
RAM access time	0.89 ns
RAM power	6.22 mW @ 444 MHz
RAM area	0.126 mm^2
ROM access time	0.74 ns
ROM power	1.13 mW @ 111 MHz
ROM area	0.024 mm^2

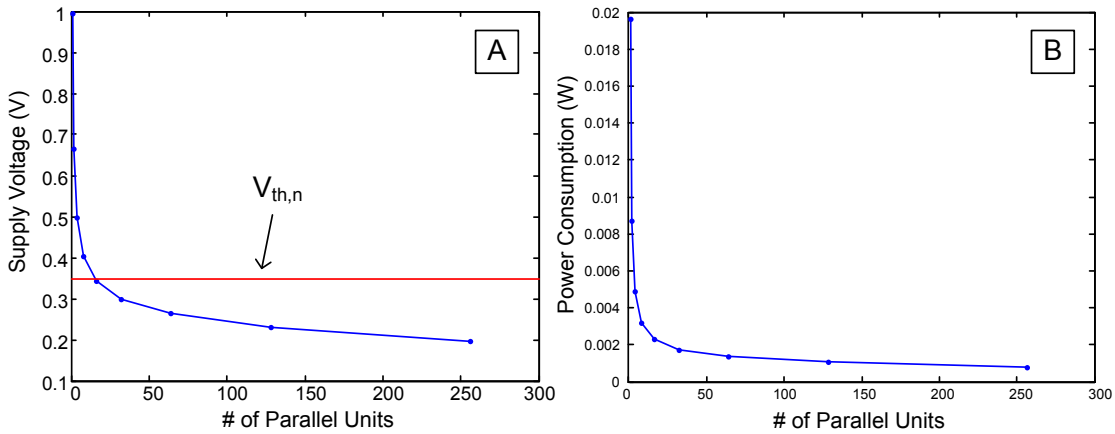


Figure 5.5: A. Supply voltage for a given number of BPs. B. Power consumption of the PE for a given number of BPs.

5.5.2 Minimum Energy Architecture Experiment Results

Figure 5.5B shows the power consumption (not including bus overhead) of the processing element(PE) for a given number of butterfly processors (BP) inside the PE. The throughput is kept constant at 222 MHz while decreasing the supply voltage. The power decreases dramatically at first, but as the supply voltage reaches the subthreshold region, there are diminishing returns for adding more elements. As the number of BPs increases, the power consumption of the bus increases due to the increased load capacitance and increased number of flip flops required for distribution and collection of data. Figure 5.6A shows the data distribution and collection overhead for a given number of butterfly processors. The power consumption of the PE decreases with additional BPs while the power consumption of the bus increases with additional BPs, so there is a point where the PE reaches its maximum power efficiency for the targeted throughput. Since the throughput is kept constant, all the power savings translate to energy savings. Figure 5.6b shows the total power consumption of the entire FFT architecture. This includes the RAM, ROM, and controller power. According to the figure, the minimum energy is achieved at $n = 32$.

5.5.3 Throughput Scaling Experiment Results

Figure 5.7A shows the results of the throughput scaling in terms of power consumption. The clock scaling and the supply voltage scaling methods are identical at 100 % duty cycle operating at nom-

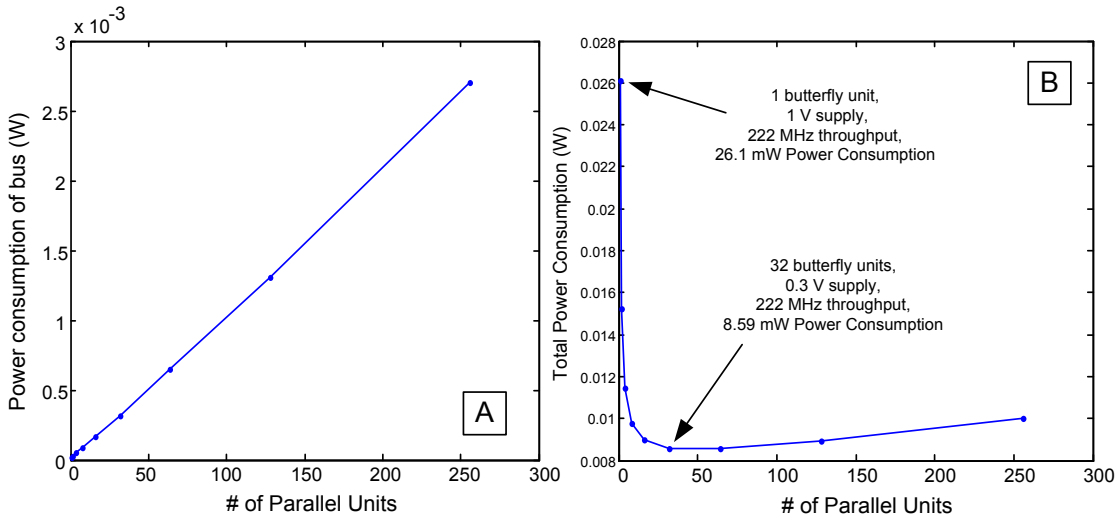


Figure 5.6: A. Power consumption of the data distribution and collection bus within the PE. B. Total power consumption of the entire architecture for a given number of BPs.

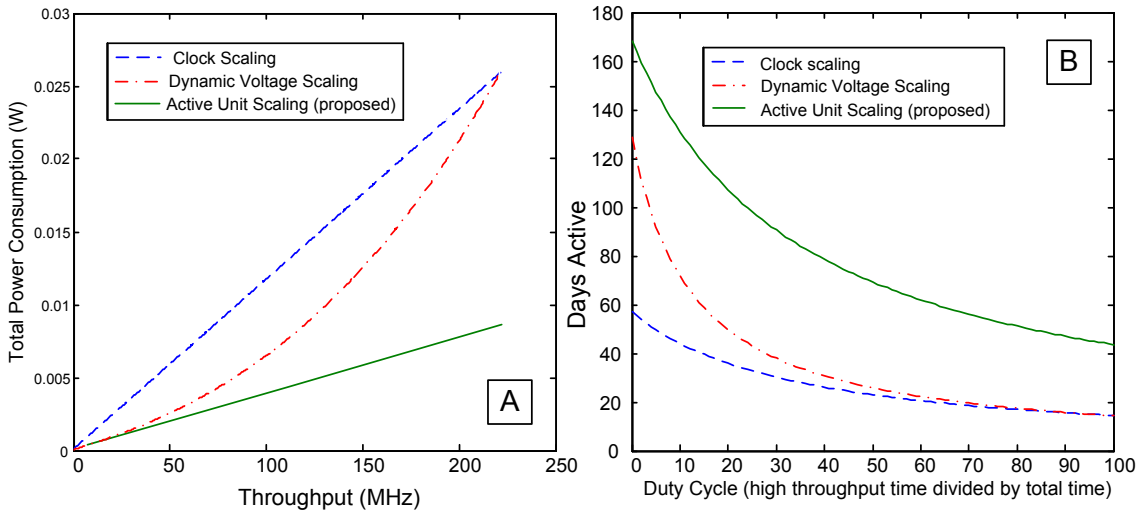


Figure 5.7: A. Total power consumption of proposed scalable-throughput design compared to a single-butterfly design. B. Number of days designs can remain active on 2 AA batteries with respect to duty cycle.

inal voltage of 1V and running at maximum speed of 222 MHz. However, at lower duty cycles, the supply voltage scaling does better than clock scaling as it reduces the power quadratically. Meanwhile, our proposed active unit scaling method outperforms both traditional methods in all cases. Figure 5.7B shows the results of the experiment mentioned in Section 5.4.2, where the throughput is varied between a 256 point FFT and a 1024 point FFT at various duty cycles. Figure 5.7B presents the number of days our design, as well as two other traditional designs, can survive while running off two heavy duty Alkaline AA batteries. The proposed active unit scaling architecture

lasts longer than the other two architectures for all possible duty cycle scenarios. As shown in Figure 5.7B, at around 30 % duty cycle, our proposed architecture using active unit scaling can last up to 3 months while the other two methods can barely survive beyond a month.

As mentioned in Section 5.3.1, the ideal method of throughput scaling is to adjust both the supply voltage and the number of active BPs. In other words, to achieve maximum energy efficiency while scaling the throughput, one has to identify the optimum number of BPs and their operating voltage for the desired throughput as shown in Section 5.5.2. Instead, we propose a simple method to simply turn off a pre-calculated number of BP's to achieve the same result. Table 5.3 shows the proposed active unit scaling vs. the ideal method of throughput scaling at 28 MHz, or one eighth of the maximum throughput. It is shown that the proposed method of throughput scaling only consumes 1.3 % more power than the ideal method. In addition, the ideal throughput scaling method involves more control logic and fine tuning of the supply voltage. The proposed scaling method, on the other hand, only requires that BPs be shut down without changing the supply voltage.

Table 5.3: The comparison between ideal throughput scaling and our active unit scaling to achieve 28Mhz throughput.

	Active BPs	Supply (V)	Processing Power (mW)
Ideal Throughput Scaling	8	0.264 V	1.153 mW
Active Unit Scaling	4	0.3 V	1.168 mW

5.5.4 Comparisons

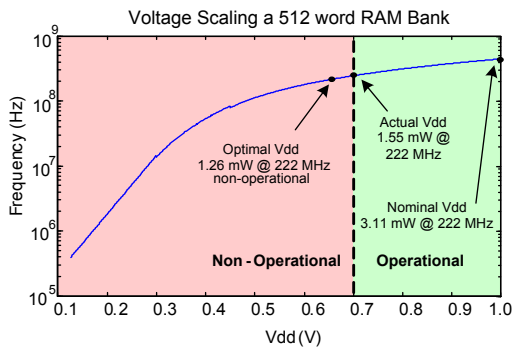
Table 5.4 compares the non parallelized FFT architectures with two parallelized FFT architectures. While $n = 32$ constitutes the minimum energy architecture, the $n = 16$ design provides a sweet spot with a slight increase in power and half of the area overhead. The total power consumption includes that for processing, RAM, ROM, and the bus.

5.5.5 Splitting the RAM Bank

Figure 5.8 shows how the frequency of a 512-word 32-bit RAM bank scales with the supply voltage, which results in significant power savings. However, in practice, it is known that regular 6T

Table 5.4: Comparison of different number of BPs

	$n = 1$	$n = 16$	$n = 32$
Total Power @ 222MHz	26.11 mW	9.00 mW	8.54 mW
Total Area	0.200 mm^2	0.947 mm^2	1.74 mm^2
PE Supply Voltage	1.0 V	0.344 V	0.30 V
Days on 2 AA bat. @ Max Duty Cyc.	14	42	44
Days on 2 AA bat. @ 15% Duty Cyc.	40 (CS), 59 (DVS)	111	118



# of Banks	Word Size per Bank	Optimal Supply Voltage	Actual Supply Voltage	Total RAM Power	Total RAM Area
1	1024	1.0 V	1.0 V	6.22 mW	0.124 mm^2
2	512	0.666 V	0.7 V	3.10 mW	0.156 mm^2
4	256	0.499 V	0.7 V	4.60 mW	0.196 mm^2

Figure 5.8: Splitting the RAM bank, parallelizing it, and reducing the supply voltage.

SRAM cells are not able to function below 0.7 V in 90 nm technology [27][2]. Therefore, even though splitting the RAM bank ideally calls for lowering the voltage, we cannot go beyond 0.7 V. The table in Figure 5.8 shows the result of our analysis. It can be seen that given the restriction on lowering the supply voltage, splitting the RAM into two banks provides the most power efficient design by reducing the memory power consumption by more than half while incurring little area overhead.

5.6 Conclusion

In this chapter, we presented a novel FFT architecture based on the traditional butterfly-based FFT architecture, and greatly reduced its power consumption by exploiting parallelism. Additional butterfly processors were added, and their supply voltage was reduced while keeping the overall throughput constant. We showed that the optimum operating voltage of butterfly processors is 0.3 V, which is in the subthreshold region. The RAM module of the design, however, was operated in superthreshold to keep the SRAM cells functional, resulting in a hybrid super/subthreshold design.

[2]Currently, there are designs of SRAM cells that are functional near or below threshold voltage. However, they require additional transistors resulting in some additional power consumption. [3]

By exploiting the parallelism in the processing element of the architecture, we were able to achieve 68 % reduction in total power consumption of the FFT architecture. A 10 % reduction is obtained by splitting the SRAM bank into two banks and scaling their voltage from nominal 1 V to 0.7 V. This total reduction of 78 % is obtained at the expense of increasing the area of the design by about a factor of 5. Even though the area increase may seem significant, the reduced cost of silicon and the increased need for low power applications such as wireless sensor networks promotes such design directions.

In addition, we proposed an efficient method to enable scaling the throughput of our design by turning off some of the active subthreshold parallel units without changing their supply voltage. We compared our method to traditional methods of clock scaling and dynamic voltage scaling by simulating the designs running on two AA batteries. Over the entire range of possible duty cycles of high throughput to low throughput, the active unit scaling outlasted the other two methods by a wide margin. At 15 % duty cycle, the active unit scaled design lasted 111 days, compared to 59 and 40 days for dynamic voltage scaling and clock scaling respectively. Future work would include fabricating a complete ASIC design of our proposed FFT architecture. This will provide us with the realistic restrictions imposed on our design, which will help us to make refinements and enhancements to it. Moreover, we intend to develop a general model for estimating how various architectures will benefit from the minimum energy architecture and active unit scaling methods detailed in this chapter.

6 Conclusion and Future Work

The focus of this thesis was to reduce the power consumption of digital signal processing (DSP) applications. DSP applications were classified into two categories. Performance constrained (PC) DSP systems require both low power and have a performance or throughput requirement that must be met. Non-Performance constrained (NPC) DSP systems have a very tight energy budget and have little to no performance requirements. With NPC systems, often performance can be sacrificed in order to reduce power. One popular method for reducing power is voltage scaling which, combined with frequency scaling, is especially effective because reducing the supply voltage of a CMOS circuit reduces the active and leakage power polynomially, while only reducing the frequency linearly. Traditionally, the limit to voltage scaling has been about half of the nominal voltage, due to incorrect operation of certain components like SRAM and phased-locked loops at lower voltages, an increased sensitivity to process variation, and a large slowdown of the circuit at lower voltages. Recently, however, it has been shown that using custom digital components designed to operate at very low voltages, it is possible to operate a CMOS circuit at a supply voltage that is lower than the threshold voltage of the transistors, in the *subthreshold region*. The circuit is more sensitive to process variation and is very slow in the subthreshold region, but the energy reduction is extremely large. For NPC applications with little to no performance constraints, the process variation and speed reduction can be tolerable. For PC applications with tighter performance requirements, subthreshold operation in combination with architectural level techniques such as pipelining and parallelization can be used to both reduce power consumption, and meet throughput requirements.

This thesis investigated the use of parallelization and subthreshold operation to lower the power consumption of DSP applications, while still meeting throughput requirements. Using an off the shelf FFT architecture, it was shown that through parallelization and subthreshold operation, a 70 % reduction in power was achieved, all while matching the performance of a nominal voltage single core architecture. With a Discrete Wavelet Transform architecture that was specifically designed to eliminate the need for SRAM banks, a power reduction of 26x was achieved compared to a reference nominal voltage architecture. Issues such as serial to parallel data distribution, dynamic throughput scaling, and memory usage were also explored. Finally, a simulation framework was presented that can characterize subthreshold circuits accurately using only fast gate-level EDA tools. This is desirable because voltage scaling greatly increases the design space and simulation times are long due to the circuit being much slower and having high accuracy requirements for the very low subthreshold current levels.

6.1 Future Work

Subthreshold operation is a relatively new field with a number of unexplored facets. Much of the ground-breaking work on subthreshold operation has been at the circuit level, but the micro-architectural level and system level concepts have not been looked into in detail. At the micro-architectural level, it has been shown in previous work [20] and in this thesis that parallelization and pipelining are both effective at obtaining moderate performance from subthreshold circuits. What has not been investigated is the limits of parallelization. There are some very high throughput systems, in data centers, switching networks, or communications systems, for example, that can greatly benefit from low power hardware. The FFT architecture in this thesis achieved 200 MHz with 16 subthreshold cores and a 70 % reduction in power consumption. Consider a system though that needs to achieve a throughput of 2 GHz. It needs to be investigated whether it is practical, for example, to have 160 parallel cores delivering a much higher throughput.

At the system level, the role between processor and memory has not been adequately inves-

tigated, especially for application specific circuits. Subthreshold memory has been thoroughly investigated, but what is unclear is the role it plays in the system level. The DWT architecture presented in this thesis was able to achieve a much higher power savings by designing hardware that eliminated the need for storing intermediate values in SRAM. Further work needs to be done, however, to determine what role memory should have in subthreshold design, and whether it is always appropriate to try to avoid as many memory accesses as possible.

Another system level aspect that needs to be investigated is the role of increased noise sensitivity. The increased noise sensitivity may mean that redundant hardware is needed in order to detect noise errors and soft errors. Redundant hardware may limit the power savings of subthreshold operation, so the tradeoff between power savings and data integrity needs to be examined. Also, process variation has a much larger impact in the subthreshold region. System level steps may need to be taken to counteract this, including binning the result chips by power consumption or performance.

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks (Amsterdam, Netherlands: 1999)*, 38(4):393–422, 2002.

[2] B. H. Calhoun and A. Chandrakasan. Characterizing and modeling minimum energy operation for subthreshold circuits. In *ISLPED '04*, pages 90–95, New York, NY, USA, 2004. ACM.

[3] B. H. Calhoun and A. Chandrakasan. A 256kb sub-threshold SRAM in 65nm CMOS. In *Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers. IEEE International*, pages 2592–2601, Feb. 2006.

[4] U. California. Bsim3v3.2 spice mos device models, 1997. `http://www-device.eecs.berkeley.edu/~bsim3`.

[5] A. Chandrakasan, R. Amirtharajah, J. Goodman, and W. Rabiner. Trends in low power digital signal processing. In *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, volume 4, pages 604–607, Monterey, CA, USA, May/June 1998.

[6] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, Norwell, MA, USA, 1995.

[7] C. Christopoulos, A. Skodras, and T. Ebrahimi. The JPEG2000 still image coding system: an overview. *IEEE Transactions on Consumer Electronics*, 46(4):1103–1127, Nov. 2000.

[8] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[9] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 36(5):961–1005, Sept. 1990.

[10] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.

[11] R. G. Dreslinkski, B. Zhai, T. Mudge, D. Blaauw, and D. Sylvester. An energy efficient parallel architecture using near threshold operation. In *PACT '07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, pages 175–188, Washington, DC, USA, 2007. IEEE Computer Society.

[12] W. R. Heinzelman, A. Sinha, A. Wang, and A. P. Chandrakasan. Energy-scalable algorithms and protocols for wireless microsensornetworks. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, volume 6, pages 3722–3725, Istanbul, Turkey, 2000.

[13] M. B. Henry, S. Griffin, and L. Nazhandali. Fast simulation framework for subthreshold circuits. In *Circuits and Systems, 2009. ISCAS '09. Proceedings of the 2009 IEEE International Symposium on*, Taipei, Taiwan, 2009.

[14] M. B. Henry, S. I. Haider, and L. Nazhandali. A low-power parallel design of discrete wavelet transform using subthreshold voltage technology. In *CASES '08: Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, pages 235–244, New York, NY, USA, 2008. ACM.

[15] M. B. Henry and L. Nazhandali. Hybrid super/subthreshold design of a low power scalable-throughput FFT architecture. volume 5409 of *Lecture Notes in Computer Science*, pages 278–292. Springer Berlin / Heidelberg, 2009.

[16] C. V. Loan. *Computational frameworks for the fast Fourier transform*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[17] J. T. Ludwig, S. H. Nawab, and A. P. Chandrakasan. Low-power digital filtering using approximate processing. In *Solid-State Circuits, IEEE Journal of*, volume 31, pages 395–400, Santa Clara, CA, USA, Mar. 1996.

[18] L. Nazhandali, M. Minuth, B. Zhai, J. Olson, T. Austin, and D. Blaauw. A second-generation sensor network processor with application-driven memory optimizations and out-of-order execution. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 249–256, New York, NY, USA, 2005. ACM.

[19] P. Pirsch. *Architectures for Digital Signal Processing*. Wiley, West Sussex, England.

[20] A. Raychowdhury, B. C. Paul, S. Bhunia, and K. Roy. Computing with subthreshold leakage: device/circuit/architecture co-design for ultralow-power subthreshold operation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(11):1213–1224, Nov. 2005.

[21] K. C. B. Tan and T. Arslan. Low power embedded extension algorithm for lifting-based discretewavelet transform in JPEG2000. *Electronics Letters*, 37(22):1328–1330, Oct. 2001.

[22] O. S. Universirty. Oklahoma state university standard cell library. `http://vcag.ecen.okstate.edu/projects/scells/`.

[23] H. Varshney, M. Hasan, and S. Jain. Energy efficient novel architectures for the lifting-based discrete wavelet transform. *IET Image Processing*, 1:305–310, Sept. 2007.

[24] A. Wang and A. Chandrakasan. A 180mv FFT processor using subthreshold circuit techniques. In *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*, pages 292–529, Feb. 2004.

[25] N. Weste and D. Harris. *CMOS VLSI Design*. Addison Wesley, third edition.

[26] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits, Vol. 31(5):677-688*, 1996.

[27] M. Yamaoka, N. Maeda, Y. Shinozaki, Y. Shimazaki, K. Nii, S. Shimada, K. Yanagisawa, and T. Kawahara. Low-power embedded SRAM modules with expanded margins for writing. In *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, pages 480–611, Feb. 2005.

[28] B. Zhai, R. G. Dreslinski, D. Blaauw, T. Mudge, and D. Sylvester. Energy efficient near-threshold chip multi-processing. In *ISLPED '07*, pages 32–37, New York, NY, USA, 2007. ACM.