Low-Energy Radix-2 Serial and Parallel FFT Designs

by

DongNi Zhang

B.A.Sc. in Electrical Engineering
University of Waterloo, 2010

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

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2013

© Massachusetts Institute of Technology 2013. All rights reserved.

Author.

Department of Electrical Engineering and Computer Science
August 30, 2013

Certified by...........................................

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

Accepted by.................................

Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Students
Low-Energy Radix-2 Serial and Parallel FFT Designs

by

DongNi Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on August 30, 2013, in partial fulfillment of the
requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

Abstract

The push for portable electronics for communication and biomedical applications has accelerated the growing momentum for high performance and low energy hardware implementations of the Fast Fourier Transform (FFT). This work presents several new hardware implementations of the radix-2 FFT algorithms that take advantage of intermittent data and parallelism to reduce the energy per FFT.

In the modified serial design, by using a low-power control memory and a pipelined data look-ahead controller to optimize processing of sequences of data with zeros, up to 45% of energy savings are achieved as compared to the baseline design. Two fully parallel FFTs with different datapaths are also developed based on a FFT flow diagram with the same geometry in each stage. Energy savings of up to 90% (an order of magnitude) are achieved as compared to the baseline design. These results are demonstrated through post-layout and parasitic extraction Nanosim simulations with 90nm standard cell libraries.

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

My journey at MIT throughout the past three years has truly been one of a kind. With family, health, and all the aspects of life packaged onto one roller coaster ride, I am truly thankful for the people in my life that made my years at MIT deeply memorable.

First and foremost, I want to thank my research advisor, Professor Anantha Chandrakasan, for taking me into his group, for being so sensitive and understanding of my situation and needs, for providing me with invaluable research advice and guidance, and for having faith and confidence in me throughout the past year. Throughout a few research glitches where I panicked, he never failed to assure me with “please don’t worry”. And while I now realize that his technical intuitions are always right, he was always humble and encouraged me to seek what I thought was right, even if I knew it conflicted with his predictions. I am very honored and grateful to have had Prof. Chandrakasan as my advisor. Without his motivation, inspiration, kindness, and patience, this thesis would not have been possible.

I also remain forever indebted to a few other professors, Professor Joel Dawson, Professor David Perrault, and Professor David Nairn (from the University of Waterloo). They not only taught me invaluable RF/power
electronics/analog circuit fundamentals, but have also become friends that I can confide in over the years.

Next, I want to express my sincere gratitude towards my colleague Rui Jin, for his immense research contributions to the modified serial FFT architecture proposed in this thesis. His understandings of the existing design, research and analysis of the improved design, as well as proficiencies and resourcefulness with tools, especially Cadence Encounter tools, have made the implementations of multiple power domain designs possible for this research.

Additionally, I want to give my sincere appreciations to Nathan Ickes, Joyce Kwong, Yildiz Sinangil, and Gilad Yahalom for taking the time out of their busy schedule to allow me to consult with them on the various aspects of the designs. I am also grateful to Margaret Flaherty and the rest of the research group for sources of inspiration and the never-dull moments in the lab.

I have been fortunate to have met many friends here at MIT whom encouraged me along the way and made my everyday worthwhile. Pleading paucity of space, I want to give a few special shout-outs. Bonnie Lam, thank you for all of our fun times together shopping, going to free food events, and hanging out in each other's dorms. Bernhard Zimmermann, remember all the late nights finishing problem sets together over bowls of fruits, smoothies, and the lamp? I am glad we balanced it out with going sailing and exploring Boston together. Maria de Soria, I had so much fun hanging out with you, cooking fish
together over Champaign, and talking about boys. Debb Hodges-Pabon, I will forever remember all our sacred tree walks together. Gilad Yahalom, Sushmit Goswami, Muyiwa Ogunnka, Mandy Woo, Kendall Nowocin, Georgia Lagoudas, and Annie Chen, you guys are the best! I cherish all of you very much, and I look forward to our life-long friendships.

Last but certainly not least, I want to thank my most loving parents, Jinying Guo and Zheng Zhang, and the love of my life, Adam Bray, for giving me endless love, support and encouragement for as long as I can remember. I know that I can always lean on you when I am beaten to the ground. You are the source of my strength and happiness in life, and I owe my life to you.
# Table of Contents

Abstract ................................................................................................................ 3

Acknowledgements ............................................................................................... 5

Table of Contents .................................................................................................. 9

List of Figures ....................................................................................................... 13

List of Tables ....................................................................................................... 15

1. Introduction ...................................................................................................... 17
   1.1 Motivation ................................................................................................. 17
   1.2 Thesis Outline .......................................................................................... 18
   1.3 Contributions ........................................................................................... 19

2. Background ....................................................................................................... 21
   2.1 FFT Algorithms ........................................................................................ 21
   2.2 Figures of Merit ......................................................................................... 24
   2.3 Previous Work .......................................................................................... 26

3. Baseline Serial Radix-2 FFT Architecture ..................................................... 31
   3.1 State Machine .......................................................................................... 32
   3.2 Memory Partitioning ............................................................................... 34
   3.3 Address Generation ............................................................................... 35
   3.4 Datapath ................................................................................................. 36
4. Proposed Radix-2 FFT Architectures ............................................................ 37

4.1 Modified Serial Architecture .................................................................. 37

4.1.1 State Machine .................................................................................. 39

4.1.2 Control Memory .............................................................................. 40

4.1.3 Controller ...................................................................................... 41

4.2 Parallel Architecture ........................................................................... 43

4.2.1 State Machine ................................................................................ 46

4.2.2 Implicit Memory ............................................................................ 47

4.2.3 Datapath ....................................................................................... 47

4.3 Modified Parallel Architecture ......................................................... 48

4.3.1 State Machine .............................................................................. 50

4.3.2 Implicit Memory ........................................................................... 51

4.3.3 Datapath ....................................................................................... 51

5. Simulation Results .................................................................................. 53

5.1 Tools and Simulation Flow ................................................................. 53

5.2 Modified Serial vs. Baseline ............................................................... 56

5.3 Parallel vs. Modified Parallel vs. Baseline ....................................... 63

6. Conclusions............................................................................................ 73

A. Alternative Forms of FFT Flow Diagrams ........................................... 77

B. Baseline Implementation Details .......................................................... 79
C. Booth Multiplier Algorithms ................................................................. 81

Bibliography ............................................................................................. 83
List of Figures

Figure 2.1: Flow Graph of the Complete Decomposition of an 8-point DFT [2]...23
Figure 2.2: Flow Graph of a Single Butterfly [2].............................................23
Figure 2.3: Source of Active Energy in CMOS Inverter [3] .........................25
Figure 2.4: Minimum-Energy Operating Point [5]...........................................28
Figure 2.5: Dynamic Voltage and Frequency Scaling (DVFS) [6] ..............29
Figure 3.1: Baseline Design Block-Diagram....................................................31
Figure 3.2: Baseline Design Finite State Machine (FSM) ...............................32
Figure 3.3: Memory Access Timing Diagram ................................................33
Figure 3.4: 8-point FFT Butterfly Sequence..................................................35
Figure 4.1: Modified Serial FFT Block Diagram ...........................................39
Figure 4.2: FFT Decimation in Time with Same Geometry in Each Stage [1]..43
Figure 4.3: Parallel FFT Block-Diagram .......................................................45
Figure 4.4: Parallel Design FSM ....................................................................46
Figure 4.5: Modified Parallel FFT Block-Diagram (8-point FFT Shown) ....49
Figure 4.6: Modified Parallel Design FSM ....................................................50
Figure 5.1: Normalized (Point-by-Point) Modified Serial FFT Simulation Results ........................................................................................................57
Figure 5.2: Normalized (Highest-Point) Modified Serial FFT Simulation Results ........................................................................................................58
Figure 5.3: Energy Breakdown of Blocks in Serial FFT Systems (Ozero) ....60
Figure 5.4: Data Memory Energy ................................................................. 61
Figure 5.5: Datapath Energy ................................................................. 61
Figure 5.6: Overhead Energy .............................................................. 62
Figure 5.7: ROM/Other Energy .............................................................. 62
Figure 5.8: Modified Serial FFT Layout and Floorplan ......................... 63
Figure 5.9: Normalized (Point-by-Point) Energy Simulation Results vs. % of zeros ......................................................... 65
Figure 5.10: Normalized (Highest-Point) Energy Simulation Results vs. % of zeros ......................................................... 67
Figure 5.11: Normalized (Point-by-Point) Energy Simulation Results vs. Performance ............................................................. 68
Figure 5.12: Normalized (Highest-Point) Energy Simulation Results vs. Performance ............................................................. 69
Figure 5.13: Parallel FFT Layout and Floorplan ........................................ 70
Figure 5.14: Modified Parallel FFT Layout and Floorplan ......................... 71
Figure A.1: FFT Decimation-in-Time with Input in Normal Order [1] .......... 77
Figure A.2: FFT Decimation-in-Time with Input and Output in Normal Order [1] ................................................................. 78
Figure A.3: FFT Decimation-in-Frequency with Input in Normal Order [1] .... 78
List of Tables

Table 2.1: Recent FFT Chips and Contributions ............................................. 30
Table 3.1: Memory Partitioning for an 8-point FFT ........................................... 34
Table 4.1: Four Variations of the Modified Serial Design Topologies ................. 42
Table 5.1: Tools, Simulation and Design Flow .................................................... 54
Table 5.2: Test Cases ........................................................................................ 55
Table 5.3: Test Cases and Parameters .................................................................. 64
Table B.1: Input Ordering vs. Address Locations of an 8-point FFT ..................... 79
Table B.2: Address Generation for an 8-point FFT ............................................. 80
Table B.3: Address Generation for an 8-point FFT ............................................. 80
Table C.1: Booth Algorithm LSB Combinations [11] ........................................ 81
Table C.2: Booth Algorithm Example 1 ............................................................... 82
Chapter 1

Introduction

1.1 Motivation

There is no shortage of signals to be sampled – audio, video, and spectrum, to name a few. The ability to interpret the acquired data through real-time computation of the Fast Fourier Transform (FFT) is the foundation for monitoring, analyzing, and controlling various systems.

The FFT is an efficient algorithm that extracts the frequency contents from a time-domain signal. It is one of the most fundamental, yet power-hungry blocks in digital signal processing because it is computationally intensive. With advancements in low-power designs, energy efficient FFT processors have become an integral part of energy-constrained applications such as modern telecommunication, sensor networks, and portable biomedical devices.
The purpose of this thesis is to present various architectural changes which further reduce the energy consumption of the existing 32-point 16-bit resolution serial radix-2 FFT engine (e.g. by Kwong in [1]), while maintaining its performance. This investigation explores the impact of parallelism and data dependency on the energy efficiency of FFT coupled with voltage scaling.

1.2 Thesis Outline

This thesis is divided into 6 main chapters. Chapter 1 introduces the motivation behind this investigation, as well as the outline and main contributions of the thesis. Chapter 2 presents relevant background concepts on FFT algorithms, along with the figure of merits. Common low-power techniques and an overview of the previous work on low-power FFT designs in the literature are also introduced. Chapter 3 dives more in-depth into the designs of the baseline serial architecture which all other designs will be based on or compared against. Chapter 4 presents the computation algorithms and design considerations for the three alternate radix-2 FFT architectures proposed. Chapter 5 touches on the tools and simulation flow used to obtain the simulation results. Additionally, a comprehensive energy comparison between the architectures for different percentage of zero-data and performance specifications are demonstrated through figures. And Finally, Chapter 6 concludes the thesis by summarizing the major findings within the thesis as well as recommending future considerations and directions for this research.
1.3 Contributions

In this thesis, three main radix-2 FFT architectures are proposed. Post-layout and parasitic extraction Nanosim simulations with 90nm standard cell libraries are used to demonstrate energy saving comparisons.

The first is a modified serial design that utilizes techniques such as clock gating, power gating, and voltage scaling in conjunction with data prediction to reduce energy. Data with zeros are commonly found in signals. In applications such as the sensor system, intermittent signals often contain clumps of zeros. In many other signals, zeros are often scattered randomly throughout the signal. In both cases, a pre-processing controller is developed to exploit data-dependency to reduce memory access and computational datapath switching activity. By using a low-power control memory and a pipelined data look-ahead controller to optimize processing of sequences of data with zeros, up to 45% of energy savings are achieved as compared to the baseline design.

The second and third are two new parallel designs that exploit parallelism to reduce supply voltage. Two fully parallel FFTs with different datapaths are developed based on a FFT flow diagram with the same geometry in each stage. The fewer cycle count allow for clock frequency to be greatly reduced, which in turn allow for a reduction in supply voltage. Energy savings of up to 90% (an order of magnitude) are achieved as compared to the baseline design.
Chapter 2

Background

2.1 FFT Algorithms

Before diving into the hardware implementation of the FFT cores, a brief
FFT background theory is given in this section. A continuous-time signal $x[t]$ can be sampled at intervals of $T_s$ to create a discrete time-domain sequence $x[n]$.

$$x[n] = x[nT_s], \quad -\infty < n < \infty \quad (2.1) \ [2]$$

$x[n]$ can be uniquely mapped to a continuous and periodic frequency-domain representation $X(e^{j\omega})$ called the Discrete-Time Fourier Transform (DTFT).

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}, \quad \omega \text{ periodic every } 2\pi \text{ interval} \quad (2.2) \ [2]$$
Sampling the DTFT of a finite $N$-length $x[n]$ produces an $N$-point discrete and periodic frequency-domain representation $X[k]$ called the Discrete Fourier Transform (DFT).

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad W_N = e^{-j(2\pi/N)}, \quad k = 0, 1, ..., N - 1 \quad (2.3)$$

Assuming the number of arithmetic multiplications and additions is a measure of computational complexity, the Fast Fourier Transform (FFT) is an efficient algorithm for computing the DFT [2]. As a comparison, for a complex $x[n]$, the direct evaluation of the DFT through Eq. (2.3) requires a total of $N^2$ complex multiplications and $N(N - 1)$ complex additions. In contrast, a radix-2 FFT algorithm requires only $\frac{N}{2} \log_2 N$ complex multiplications and $N \log_2 N$ complex additions [2].

This is due to the fact that FFT algorithms are based on recursively decomposing the DFT of an $N$-length sequence into smaller-length DFT that are then combined together. Figure 2.1 shows the flow graph of the complete decomposition for an 8-point DFT. Appendix A lists a few alternative forms of the FFT flow diagram. A fully decomposed DFT is referred to as a radix-2 FFT, as the computation is reduced to that of 2-point DFT. The 2-point DFT block is referred to as a butterfly and is shown in Figure 2.2. In a radix-2 $N$-point FFT, there are $N \log_2 N$ stages, with each stage containing $\frac{N}{2}$ basic blocks.
Figure 2.1: Flow Graph of the Complete Decomposition of an 8-point DFT [2]

\[ x[0] \rightarrow W_N^0 \rightarrow X[0] \]
\[ x[4] \rightarrow W_N^0 \rightarrow -1 \rightarrow X[4] \]
\[ x[2] \rightarrow W_N^0 \rightarrow W_N^2 \rightarrow X[2] \]
\[ x[6] \rightarrow W_N^0 \rightarrow -1 \rightarrow W_N^2 \rightarrow X[6] \]
\[ x[1] \rightarrow W_N^0 \rightarrow -1 \rightarrow X[1] \]
\[ x[5] \rightarrow W_N^0 \rightarrow -1 \rightarrow X[5] \]
\[ x[3] \rightarrow W_N^0 \rightarrow -1 \rightarrow W_N^2 \rightarrow X[3] \]
\[ x[7] \rightarrow W_N^0 \rightarrow -1 \rightarrow X[7] \]

Figure 2.2: Flow Graph of a Single Butterfly [2]

\[ X_{m-1}[p] \rightarrow X_m[p] \]
\[ X_{m-1}[q] \rightarrow W_N^r \rightarrow -1 \rightarrow X_m[q] \]

\[ X_m[p] = X_{m-1}[p] + X_{m-1}[q]W_N^r \] \hspace{1cm} (2.4a) [2]
\[ X_m[q] = X_{m-1}[p] - X_{m-1}[q]W_N^r \] \hspace{1cm} (2.4b) [2]

For each butterfly computation, a pair of values is taken from the \((m - 1)^{th}\) stage and a new pair of values is computed for the \(m^{th}\) stage. The equivalent equations being computed are found in Eq. 2.4.
2.2 Figures of Merit

To quantitatively evaluate and compare various hardware implementations of the FFT algorithm, a set of figures of merit are introduced here.

The first metric is performance. In this investigation, the performance \( t_{\text{one FFT}} \) is defined as the total time taken to finish computing one FFT. So for a given performance specification, the minimum clock frequency \( f_{\text{clk, min}} \) can be calculated by Eq. (2.5) if the number of cycles needed to perform the calculation, \( n_{\text{cycles taken}} \), is fixed for an architecture.

\[
T_{\text{clk,max}} = \frac{t_{\text{one FFT}}}{n_{\text{cycles taken}}} \rightarrow f_{\text{clk, min}} = \frac{n_{\text{cycles taken}}}{t_{\text{one FFT}}}
\]  

(2.5)

For a given process, there is a maximum supply voltage \( V_{DD} \) which the design can operate at, which sets a lower bound for the propagation delay. Depending on the design architecture, if the critical path is too long, there may not exists a clock frequency that satisfies both Eq. (2.5) and the setup and hold time constraints, in which case the design simply cannot meet performance. With performance requirement as a variable, this investigation compares various designs for different performance specifications.

The second metric is energy consumption per FFT, with main focus on the dynamic (or switching) energy and the static (leakage) energy in this work.
The active energy $E_{\text{dyn}}$, as shown in Figure 2.3, is due to charging of load and parasitic capacitances when switching from 0 to 1, and can be calculated by Eq (2.6). As an observation, lowering the physical capacitance $C_L$ or reducing the switching activity (or probability of switching from 0 to 1 $P_{0\rightarrow1}$) both have proportional effects on reducing active energy. Meanwhile, lowering the supply voltage $V_{DD}$ has a quadratic effect on active energy [3].

$$E_{\text{dyn}} = C_L V_{DD}^2 P_{0\rightarrow1}$$  \hspace{1cm} (2.6) [3]

![Figure 2.3: Source of Active Energy in CMOS Inverter [3]](image)

The leakage energy $E_{\text{stat}}$, is due to current flowing through the reverse-biased diode junctions of the transistors even when the transistor is “off” [4]. Eq (2.7b) calculates the sub-threshold current. For an inverter, the leakage energy is calculated by evaluating Eq (2.7a) at $V_{GS} = 0V$. As an observation, lowering the device’s threshold voltage $V_T$ (or stacking the devices) has an exponential effect on reducing the leakage energy.
\[ E_{stat} = V_{DD} \int_{t=0}^{t_{one \, FFT}} i_{sub}(t) \, dt \]  \hspace{1cm} (2.7a) [4]

\[ I_{sub} = I_o e^{\frac{V_{GS}-V_T+nV_{DS}}{nV_{th}} \left( 1 - e^{-\frac{V_{DS}}{V_{th}}} \right)} \]  \hspace{1cm} (2.7b) [4]

Other metrics of less importance for the purpose of this investigation include the cost or area of the chip, the resolution (which is fixed at 16-bit), and the number of points \( N \) (which is fixed at 32-point) for the designs discussed in this thesis.

In general for any hardware design, a tradeoff exists between performance and energy. In the context of this quest, the goal is to achieve lower energy per FFT for a given performance specification. This is different from the goal of minimizing energy regardless of performance.

### 2.3 Previous Work

This section will give a brief overview on the common low-power design techniques and the achievements of other scholars in the field of low-energy FFT processor design. Clock gating, power gating and voltage scaling are some of the common techniques used in low-power designs.

Clock gating is a technique that adds more logic to prune the clock tree. By disabling portions of the circuit flip flops from switching state when disable
conditions are set, switching power consumption goes to zero, and only leakage power are incurred [12].

Power gating is a technique that shuts off the supply current to blocks of the circuit via switching transistors placed at header and/or footer. By stacking transistors during system stand-by, the leakage power can be reduced [4]. However, entering and exiting the sleep mode safely also increases timing delays. Moreover, architectural trade-off exists between leakage energy saved in stand-by mode and the energy consumed to enter and exit the sleep mode [4]. Power gating methods such as the fine-grain power gating, coarse-grain power gating are sometimes used, and will not be covered in details here. Other cells of importance often used in power gating circuitry include the isolation cells and retention registers. Isolation cells are usually placed in between the power gated block and the normally-On block the first is driving. This prevents the short circuit current in the normally-On block caused by its floating input when the first block is put to sleep [13]. Retention blocks are usually composed of low leakage flip flops used to hold the data when entering sleep and restore the system state when the system is reawakened [13].

Dynamic voltage scaling is a technique that increases or decreases the supply voltage to a block depending on the circumstances. Increasing the supply voltage is sometimes used to increase the performance of a circuitry, as it causes capacitances to be charged and discharged quicker [3]. Decreasing the supply voltage is sometimes used to save power, as the switching power
dissipated by static CMOS gates decreases quadratically with decreasing voltage (Eq. 2.6).

Many previous works looked at optimizing energy for the serial radix-2 algorithm portrayed in Figure 2.1. Wang's work looked at lowering the supply voltage $V_{DD}$ to the minimum energy point in order to save the total energy. The minimum energy point is the point at which the total energy is at an absolute minimum, without considering any performance constraints. Figure 2.4 demonstrates that the minimum energy point for this FFT system implemented in a 0.18um process corresponds to a supply voltage of 380mV, which results in a 13kHz operating frequency [5].

![Figure 2.4: Minimum-Energy Operating Point [5]](image-url)
In certain FFT applications, however, a minimum throughput is needed such that the supply voltage cannot be fixed at the minimum energy point. In these cases, the supply voltage may be dynamically adjusted based on the processing needs of the current input data in order to meet performance. Figure 2.5 exhibits a dynamic voltage scaling technique proposed in [6], in which the supply voltage is adjusted between FFT computations based on the timing of a replica critical path. Since the voltage is adjusted only once per FFT, it must satisfy the timing of the longest-delay iteration within the FFT computation. This means the supply voltage chosen for each FFT must accommodate the iteration that requires the most processing, thus energy is wasted during shorter-delay iterations.

![Diagram of Dynamic Voltage and Frequency Scaling (DVFS)](image)

**Figure 2.5**: Dynamic Voltage and Frequency Scaling (DVFS) [6]

Table 2.1 tabulates a summary of the previous recent FFT chips and their contributions.
### Table 2.1: Recent FFT Chips and Contributions

<table>
<thead>
<tr>
<th>Author, Year</th>
<th>Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice Wang, 2005</td>
<td>Sub threshold logic and memory are developed for a radix-2 serial FFT processor in 0.18um standard CMOS process. Minimum energy point for 16-bit 1024-point FFT is found to occur at a supply voltage of 350mV, where it dissipates 155nJ/FFT at clock frequency of 10kHz. [5]</td>
</tr>
<tr>
<td>Nathan Ickes, 2008</td>
<td>A different control scheme for ordering the butterflies in a radix-2 serial FFT processor is developed to reduce stalls and enhance performance. [7]</td>
</tr>
<tr>
<td>Yuan Chen, 2008</td>
<td>A multimode multipath-delay-feedback architecture is proposed for a parallelized radix-2 FFT processor, fabricated in UMC 90nm single-poly nine-metal CMOS process. DVFS techniques are used to save power by 18% - 43% depending on the operation mode. [6]</td>
</tr>
<tr>
<td>Mingoo Seok, 2011</td>
<td>A super pipelining architecture modification is proposed which allows shortening of clock period and further voltage scaling to improve energy efficiency. The 1024pt complex FFT operating at 30MHz and 0.27V consumes only 17.7nJ of energy. [8]</td>
</tr>
<tr>
<td>Joyce Kwong, 2011</td>
<td>A different control scheme for sequencing the butterflies in a radix-2 serial FFT processor is developed which reduces switching activity and active energy. The control scheme reduces the datapath power by 50% compared to a reference design. [1]</td>
</tr>
<tr>
<td>This work, 2013</td>
<td>A data-aware input-adaptive architecture is proposed for a serial radix-2 FFT processor. This work dynamically optimizes energy based on the workload of each butterfly within an FFT by using a controller that looks ahead at the data sequences. Through further minimization of switching activity (reducing dynamic energy), power gating multiple power domains (reducing leakage energy), and voltage scaling memory (reducing total energy), up to 50% energy savings was achieved compared to [1]. Additionally, two parallel radix-2 FFT processors are proposed. Through parallelism, further lowering of supply voltage lead to savings of up to 90% compared to [1].</td>
</tr>
</tbody>
</table>
Chapter 3

Baseline Serial Radix-2 FFT Architecture

The reference design in which all other proposed architectures are compared against is introduced in this section. The block diagram of this serial radix-2 32-point complex-valued FFT implementation is given in Figure 3.1 below.

Figure 3.1: Baseline Design Block-Diagram
This design is directly based on Joyce Kwong's FFT implementation found in [1], with truncation from 512-point down to 32-point to shorten power-simulation time. In addition, flip-flop based memory is used in the simulation instead of SRAM due to lack of access for this specific process.

### 3.1 State Machine

The 32-point serial implementation is based on the FFT algorithm in Figure 2.1, where there are $\frac{32}{2} = 16$ butterflies down in every stage and $\log_2 32 = 5$ stages across, yielding a total of 80 butterflies. For this in-place computation, the different nodes on the same horizontal line represent the same single memory location that is updated through time.

The baseline system in Figure 3.1 operates in three stages: load, compute, and unload, as illustrated by Figure 3.2 below.

![Figure 3.2: Baseline Design Finite State Machine (FSM)](image)

In the first stage, a time-domain input (real and imaginary) is written into the data memory every clock cycle, until all 32 inputs are loaded.
In the second stage, the processor steps through one butterfly every clock cycle, until all 80 butterflies are computed. For every clock cycle, two inputs A and B are fetched from the memory and manipulated in the butterfly datapath, while at the same time the manipulated X and Y from the previous cycle are written back to their original locations to be available as inputs for future butterfly iterations. The timing diagram in Figure 3.3 clarifies this.

![Figure 3.3: Memory Access Timing Diagram](image)

Upon completing all the butterflies, the system enters the third stage, where data is read out from the memory every clock cycle, until all 32 memory locations are unloaded. This completes the FFT as the outputs are the frequency-domain results.
3.2 Memory Partitioning

For a 32-point FFT, 32 address locations are needed, with each location storing a real value and an imaginary value. The time-domain FFT inputs are stored in bit-reverse order, as explained in Appendix B.

A complete butterfly is computed every clock cycle, which requires the ability to simultaneously perform two reads and two writes every clock cycle. The memory was thus split into four memory banks based on the most significant bit (MSB) and parity of each address [7], to avoid memory access collision. Table 3.1 illustrates the memory partitioning for an 8-point FFT.

<table>
<thead>
<tr>
<th>FFT Input Order</th>
<th>Memory Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr</td>
<td>input</td>
</tr>
<tr>
<td>000</td>
<td>x[0]</td>
</tr>
<tr>
<td>001</td>
<td>x[4]</td>
</tr>
<tr>
<td>010</td>
<td>x[2]</td>
</tr>
<tr>
<td>011</td>
<td>x[6]</td>
</tr>
<tr>
<td>100</td>
<td>x[1]</td>
</tr>
<tr>
<td>101</td>
<td>x[5]</td>
</tr>
<tr>
<td>110</td>
<td>x[3]</td>
</tr>
<tr>
<td>111</td>
<td>x[7]</td>
</tr>
</tbody>
</table>

Initially, the two addresses always have the same MSB but differ in parity, and subsequently rotate between sets with high MSB and low MSB. In the last stage, the two addresses always have different MSBs and different parity, and subsequently rotate between sets with different combinations.
3.3 Address Generation

The butterfly operations are specifically ordered so that sequential butterflies involve disjoint sets of memory banks [7], and no addresses will be read before they were written to by the previous butterfly iteration. Figure 3.4 below highlights the butterfly ordering for an 8-point FFT, with the red number indicating the clock cycle count. This ordering avoids memory collisions while also minimizing switching activity since the butterflies with the same twiddle factor $W_N$ are performed consecutively in each stage. The ROM will still be accessed every clock cycle, but the bus switching will be reduced. Note that no butterflies are performed in cycle 9 due to the stall necessary to prevent memory access collisions upon entering the very last stage.

---

Figure 3.4: 8-point FFT Butterfly Sequence
This address sequence is generalized in Appendix B for all but the last stage. In
the last stage, the addresses are generated using gray-code counters instead.
The address generated for an 8-point FFT is shown in Appendix B.

3.4 Datapath

The datapath takes two pairs of complex numbers as inputs, implements
the butterfly in Figure 2.2 in hardware through combinational logics, and
outputs two pairs of complex numbers.

To implement Eq (3.2), the datapath requires 4 two-input multipliers to
calculate $B_r \ast W_r$, $B_i \ast W_i$, $B_r \ast W_i$, and $B_i \ast W_r$, along with 4 three-input adders to
calculate $X_r$, $X_i$, $Y_r$, and $Y_i$. This design uses the built-in Design Ware
multipliers and adders.

\[
\begin{align*}
(X_r + jX_i) &= (A_r + jA_i) + (B_r + jB_i) \ast (W_r + jW_i) \\
&= (A_r + B_r \ast W_r - B_i \ast W_i) + j(A_i + B_r \ast W_i + B_i \ast W_r) \\ \\
(Y_r + jY_i) &= (A_r + jA_i) - (B_r + jB_i) \ast (W_r + jW_r) \\
&= (A_r - B_r \ast W_r + B_i \ast W_i) + j(A_i - B_r \ast W_i - B_i \ast W_r)
\end{align*}
\]
Chapter 4

Proposed Radix-2 FFT Architectures

Three architectures are proposed in this section, with the first one being a direct modification of the reference design in [1], and the latter two being the new parallel implementations. Functionally, all the designs produce the exact same FFT results as the baseline design.

4.1 Modified Serial Architecture

This section looks at dynamically optimizing energy based on the workload of each butterfly iterations within one FFT, as oppose to between FFTs. The four variations of this proposed architecture are all based on the reference design previously discussed. As noted in the Acknowledgement, the
author wants to express her gratitude towards Rui Jin for the materials in Section 4.1 and Section 5.2 of this thesis. The modified serial architecture proposed in this section, and the simulation results obtained in Section 5.2 are made possible by the immense design, implementation, and analysis contributions from Rui.

Data with zeros are commonly found in signals. In applications such as the sensor system, intermittent signals often contain clumps of zeros. In many other signals, zeros are often randomly scattered throughout the signal. In both cases, a pre-processing controller can exploit data-dependency to reduce memory access and switching activity. In the baseline design, it was observed through simulations that even when a multiplicand is zero, the changing multiplier (twiddle factor), results in significant processing delays before yielding a simple product of zero. Having prior knowledge of the nature of the incoming data and the twiddle factor allows simplification of computation and removal of glitches in the datapath. Through disabling, power-gating, and voltage-scaling different blocks, reductions in active energy (switching activity) and leakage energy can be achieved.

The block diagram of this modified serial radix-2 FFT implementation is given by Figure 4.1 below. The data-aware design uses the same memory partition and address generation as the original design. However, it has an additional pre-processing controller.
4.1.1 State Machine

This system operates in the same three stages as the baseline design: load, compute, and unload, with a few adjustments. In the first stage, while loading each of the FFT inputs, a representative 2-bit (1-bit real and 1-bit imaginary) flag of the corresponding data is also loaded into the control memory.
simultaneously. The flag represents whether the corresponding data in the memory is zero.

In the second stage, butterflies are computed in the same sequence, but with additional energy-saving look-ahead logics that pre-fetch the corresponding flags from the control memory and determine whether to read the data memory, enable the adders and multipliers in the datapath, write back to the data memory, power gate the datapath and voltage scale the data memory. There is a trade-off between energy saved by a system in sleep and the energy consumed to enter and exit the sleep state [9]. Thus, the design must consider future iterations to determine the best optimizations. Bits from the control memory are pipelined to the control logic, so that the workload of the future iterations can be predicted. Through post layout and extraction Nanosim simulations in Chapter 5, the break-even point for this design is found to be 2-cycle look-ahead. With this pipelined structure, the original throughput of one butterfly computation every clock cycle is maintained.

Upon completing all butterflies, the system enters the unaltered third stage to unload the FFT frequency-domain results from data memory.

4.1.2 Control Memory

The control memory and data memory are initially populated simultaneously using the same memory partitioning described in Section 3.2,
and thus share the same address locations and address generator for fetching
and updating data and flags during the compute phase.

A flag of "1" indicate that all bits of the corresponding data are zeros. At
every clock cycle, 5 flags are fetched and updated (written back to the control
memory): $is0_{A_r}, is0_{A_i}, is0_{B_r}, is0_{B_i}, is1_{W_r}$ (equivalent to $is0_{W_i}$). The twiddle
factor flag is pre-determined based on the butterfly iteration count. Updated
flags are computed through a much simpler butterfly logic shown in Eq (4.1),
which is derived from Eq (3.2).

$$
\begin{align*}
    is0_{X_r} &= is0_{Y_r} = is0_{A_r} \text{ AND } is0_{B_r} \text{ AND } (is1_{W_r} \text{ OR } is0_{B_i}) \\
    is0_{X_i} &= is0_{Y_i} = is0_{A_i} \text{ AND } (is1_{W_r} \text{ OR } is0_{B_r}) \text{ AND } is0_{B_i}
\end{align*}
$$

(4.1)

4.1.3 Controller

Table 4.1 tabulates the control logic design variations to be implemented
and tested for energy savings comparison. Features such as disabling Memory
read/write (Mem Dis), disabling datapath (DP Dis), memory voltage scaling
(Mem VS), and datapath power gating (DP PG) are selectively implemented in
the tree operation stages. An energy savings comparison between the four
control logic variations are found in Section 5.2. The objectives are to compare
the datapath energy savings with the data memory energy savings and to
determine the optimum look-ahead amount.
Table 4.1: Four Variations of the Modified Serial Design Topologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Look-Ahead Cycles</th>
<th>Load</th>
<th>Compute</th>
<th>Unload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mem Dis</td>
<td>DP Dis</td>
<td>Mem VS</td>
</tr>
<tr>
<td>Baseline</td>
<td></td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gated DP 1</td>
<td>1</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gated DP 2</td>
<td>2</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gated Memory 2</td>
<td>2</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gated Memory 3</td>
<td>3</td>
<td>--</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the load and unload stages, the datapath can be disabled to eliminate unnecessary switching activity and reduce active energy. Power gating the datapath can also reduce leakage energy. Unlike the compute stage, the critical path is shorter without the datapath delay so the supply voltage to the memory can be lowered from $V_{DD1}$ to $V_{DD2}$ to further reduce energy. In the compute stage, memory read and write can be disabled to save energy for the four memory locations if predicted unnecessary (butterfly manipulations that will result in no change to the data, data to be fetched is zero, etc). Memory supply voltage can once again be lowered when the memory is disabled. The four multipliers and four adders in the datapath can also be individually disabled and power gated to save energy for cases such as multiply by 1 (W0 twiddle factors), multiply by 0 (low activity input data), or add 0.

Our overall contribution is a fully working low-energy pre-processing controller not in the critical computation path that can predict the workloads of future iterations and adjust the performance of the critical path accordingly. This data-aware input-adaptive architecture is expected to save more energy as
the data becomes more intermittent or if a greater percentage of inputs are zeros. As deliverables, plots of energy consumption as activity varies are shown in Chapter 5 of the thesis.

4.2 Parallel Architecture

This section proposes a parallel FFT architecture where all butterflies in each stage are performed simultaneously. Again, a radix-2 decimation in time FFT algorithm is used. This time however, a rearrangement of Figure 2.1 is used to simplify data access. If each register continues to correspond to its horizontal position in Figure 2.1, the inputs to the butterflies would have to come from different positions for every new butterfly stages.

![Figure 4.2: FFT Decimation in Time with Same Geometry in Each Stage [1]](image)
As illustrated in Figure 4.2 above, each stage now has the same geometry, which eliminates the dynamic routing complexity otherwise needed to implement Figure 2.1. Instead of multiplexing large amount of data as inputs to the butterflies for different stages, the same connections and logics can be reused for all stages.

The block diagram of this parallel radix-2 FFT implementation is given by Figure 4.3 below. Naturally, the parallel design is not restrained by serial memory access, so the serial load and unload stages, the address generator, and the four banks of memories that were used to store the butterfly results are no longer necessary, thereby reducing energy. In this design, one set of internal registers are used to store the intermediate calculated values at every stage. The odd and even datapaths differ in that one consists of both the multi-cycle multiply stage and the add stage, while the other only consists of the add stage. The datapaths’ inputs come from hardwired locations.
Figure 4.3: Parallel FFT Block-Diagram
4.2.1 State Machine

The coarse FSM of this parallel design is shown in Figure 4.4 below. Unlike the serial architectures, time-domain inputs no longer need to take multiple clock cycles to load, and outputs of FFT no longer need to take multiple clock cycles to unload.

![Figure 4.4: Parallel Design FSM](image)

Instead, the first stage only takes one clock cycle. This stage either loads the initial FFT inputs if entered via reset, or it sets up the registers for new multiplication iterations after completion of each stage. The second stage is separated into two stages: multiplication and addition. The multiplication stage uses multiple clock cycles (cycle count equal to number of bit resolution) to perform shift-add multiplications. The add stage only takes one clock cycle. Upon iterating through all the 5 stages for a 32-point FFT, the system enters the last state, indicating the completion of the FFT. This design can easily be extended to compute an N-point Mbit resolution FFT.
4.2.2 Implicit Memory

The implicit memory consists of odd and even registers. The even registers are used to store $A$. The odd registers are much larger. These registers are used to store either $B$ or the four intermediate datapath results $B_r W_r$, $B_l W_l$, $B_r W_l$, and $B_l W_r$. From the block diagram in Figure 4.3, notice that each data is computed in-place, and directly corresponds to its position in the modified FFT flow chart in Figure 4.2. This memory structure allows the same hard-wired inputs to the datapath at each stage, and eliminates unnecessary dynamic routings. Differentiating between odd and even registers also allows for a smaller design, since datapaths for even registers don't require multipliers.

4.2.3 Datapath

The addition algorithm used for this design is straightforward, so this section will focus on the multiplication algorithm. A pipelined multi-cycle shift-add multiplier replaces the Design Ware multiplier from the serial design, and is instantiated $\frac{N}{2}$ times for the parallel design. By breaking the combinational logic down into smaller pipelined logic, the area of the design as well as the propagation delay for each clock cycle (and hence the supply voltage needed) can be reduced.
To multiply two 16-bit numbers, only 1 clock cycle is required by the Design Ware multiplier, whereas 16 clock cycles are needed by this pipelined multiplier. For each stage in the 32-point parallel FFT design, 16 butterflies are computed simultaneously, and require 16 clock cycles to complete. For each stage in the serial design, the 16 butterflies are computed sequentially, but also take 16 clock cycles to complete. A net reduction in the total number of clock cycles is still achieved by the parallel design, as loading and unloading data do not require multiple clock cycles.

The Booth multiplication algorithm used in this design is a variation of the simple shift-add algorithm. It handles multiplication of two signed numbers in 2's complement representation by iteratively examining two adjacent pairs of the multiplier's LSB every cycle [10]. Appendix C shows the actions needed for each of the four LSB combinations, and an example of the Booth algorithm.

4.3 Modified Parallel Architecture

This section proposes a modified parallel FFT architecture where the multi-cycle pipelined datapaths described in Section 4.2 are replaced with direct instantiations of the baseline single-cycle combinational datapath. A fully-parallel datapath where all the butterflies are unrolled is not considered here.

Again, the radix-2 decimation in time FFT algorithm in Figure 4.2 is used. The block diagram of this modified parallel radix-2 FFT implementation
is given by Figure 4.5 below. For simplicity, the block diagram for an 8-point FFT is shown.
As with the previous parallel architecture, this design is not restrained by serial memory access, so the serial load and unload stages, the address generator, and the four banks of memories that were used to store the butterfly results are no longer necessary. Since the datapath is no longer split into multi-cycle multiplication and add, the odd registers no longer need to be large enough to accommodate the intermediate products $B_r W_r$, $B_i W_i$, $B_r W_i$, and $B_i W_r$.

4.3.1 State Machine

The FSM of this modified parallel design is shown in Figure 4.6 below. This FSM is simpler than that of the previous parallel design FSM because the computation of a single butterfly is not split over multiple clock cycles.

![Figure 4.6: Modified Parallel Design FSM](image)

The first stage loads the initial FFT time-domain inputs and still only takes one clock cycle. In the second stage, 16 butterflies are computed in parallel every clock cycle for 5 clock cycles, until all 5 iterations of the 32-point FFT are complete. The system then enters the last state, indicating the completion of the FFT.
4.3.2 Implicit Memory

The implicit memory consists of even and odd registers used to store A and B, respectively. The outputs of these registers are directly fed into the corresponding butterfly. Each data is computed in-place, and directly corresponds to its position in the modified FFT flow chart in Figure 4.2. This memory structure allows the same inputs to the datapath at each stage, and eliminates unnecessary dynamic routings. Unlike the previous parallel design, no differentiation between odd and even registers is really necessary except for differentiating between the A and B butterfly port connections.

4.3.3 Datapath

This design uses the same datapath as that of the baseline design, only instantiated multiple times. The use of a single-cycle datapath greatly reduces the number of clock cycles needed to compute the FFT. Only $\log_2 32 = 5$ clock cycles are needed to compute the 32-point FFT, which is $\frac{1}{16}$ of that used by the previous parallel design. Given a fixed requirement on the FFT completion time, the frequency of the clock can be greatly reduced which allows the supply voltage to be scaled lower. This along with the reduction in register count all contribute to a significant reduction in energy consumption.
Chapter 5

Simulation Results

In this chapter, the simulation results will be presented to evaluate the effects of parallelism and data-dependency on energy. The tools and simulation flow are introduced first, followed by a quantitative comparison of the designs. All the simulation results are from post-layout and parasitic Nanosim simulations. The test cases include varying the percentages of zero-data for different FFT performance requirements.

5.1 Tools and Simulation Flow

Each design (baseline, modified serial (x4), parallel, and modified parallel) follow the design and simulation flow summarized in Table 5.1 below. They are carried out from RTL design through to synthesis and place and route. Matlab is used to create the golden model and to assist with the verification of
the designs. The technology used to generate the simulation results in this Chapter is the generic 90nm Cadence gpdk090 process.

Table 5.1: Tools, Simulation and Design Flow

<table>
<thead>
<tr>
<th>Flow</th>
<th>Tools</th>
<th>Outputs generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. RTL Simulation</td>
<td>Cadence NC Verilog (functional simulation tool)</td>
<td>.txt (simulation outputs) .vcd (simulation waveforms)</td>
</tr>
<tr>
<td>2. RTL Debugging</td>
<td>Cadence SimVision (viewing digital waveforms) Matlab (numerical computing environment)</td>
<td>.txt (golden model)</td>
</tr>
<tr>
<td>3. Synthesis</td>
<td>Cadence RC (synthesis tool)</td>
<td>.rpt (synthesis log files) .v (synthesized gate netlist)</td>
</tr>
<tr>
<td>4. Define Power Domains</td>
<td></td>
<td>.cpf (define power domains)</td>
</tr>
<tr>
<td>5. Place and Route</td>
<td>Cadence Encounter (place and route tool)</td>
<td>.gds (chip layout details) .v (gate netlist) Reports</td>
</tr>
<tr>
<td>6. DRC, LVS, and Parasitic Extraction</td>
<td>Cadence Virtuoso (custom IC design tool)</td>
<td>.sp (spice netlist containing parasitics)</td>
</tr>
<tr>
<td>7. Generate Input Vectors</td>
<td>Nanosim (power simulation tool)</td>
<td>.vec (input vectors for testbench)</td>
</tr>
<tr>
<td>8. Post Layout Simulation</td>
<td>Nanosim CScope (viewing digital and analog waveforms)</td>
<td>Reports (containing power info) .fsdb (simulation waveforms)</td>
</tr>
</tbody>
</table>

For each of the designs, the corresponding SPICE netlist generated in step 6 is used in conjunction with different combinations of "vec" vector files and "sp" test benches for the Nanosim simulations. The lowest power and energy associated with each design are evaluated for the matrix of test cases summarized in Table 5.2.
Covering all these test cases is an iterative process. For a given design and a given FFT completion time requirement, the associated clock frequency for that design is calculated and set in the Verilog test bench. The 0zero test case is the default, and the simulation is run in step 1 to generate the "vcd" simulation waveform. Step 7 is then performed to convert the "vcd" file into a "vec" file using Nanosim's vcd2vec command. Next, the supply voltage is set to the nominal 1.2V in the .sp test bench and then Nanosim is run in step 8 to generate the "fsdb" simulation waveform. The waveform is viewed in CScope to confirm correct functionality. To determine the lowest working supply voltage, the supply voltage is incrementally lowered and the simulation is rerun until the waveform is no-longer correct. Step 1 and 7 are then repeated for different percentages of zero-data inputs, and the various "vec" files are used with the minimum supply voltage to perform Nanosim simulations in step 8. This entire process is rerun for the different architectures. Just to reiterate, all the designs are functional and produce the same FFT results as the baseline design.
5.2 Modified Serial vs. Baseline

This section compares the post layout (with standard cell blocks) and post parasitic extraction Nanosim simulation results between the four variations of the modified serial design and the baseline design for computing a 32-point FFT. The energy comparisons are based on the test case with a fixed FFT completion time of 4280ns. For the serial design, this corresponds to a clock frequency of 50MHz and a minimum supply of $V_{DD1} = 0.6V$ and $V_{DD2} = 0.5V$. Note that due to the lack of readily available SRAM in gpdk090, flip-flop based memories are used instead. With SRAM, similar energy savings are expected.

Figure 5.1 and Figure 5.2 below show the simulation results of the energy per FFT for the four designs compared to the baseline design (with no pre-processing controller), given various proportions of input zeros. Figure 5.1 is a point-by-point normalization of the new designs to the baseline design. In this normalization scheme, for each percentage of zeros considered, the normalized energy per FFT is calculated by

$$\frac{\text{Energy per FFT for design X}}{\text{Energy per FFT for baseline \ evaluated at a given \% of zero}}.$$  

The baseline curve for this normalization is always flat at 1. The point-by-point normalization scheme provides a clear comparison and easy evaluation of the relative percentage of energy savings for various proportions of zeros. Figure 5.2 on the other hand is a normalization of the new designs to the highest energy point of the baseline design. In this normalization scheme, for each percentage of zeros considered, the normalized energy per FFT is calculated by
Energy per FFT for design X, evaluated at a given % of zeros. Maximum energy per FFT for baseline.

Different arrangements of zeros within the inputs, including clumps of zeros at the beginning of an FFT input stream, clumps of zeros at the end, and random zeros throughout, are analyzed. They all give very similar energy results. The Nanosim results were validated through correlating datapath energy with SPICE results.

Figure 5.1: Normalized (Point-by-Point) Modified Serial FFT Simulation Results
Figure 5.2: Normalized (Highest-Point) Modified Serial FFT Simulation Results

The four design variations are defined in Table 4.1. The “gated DP (1)” design has control logics that can disable the data memory, datapath, or ROM, and power gate the datapath based on the workload on each clock cycle. Second, the “gated DP (2)” design has control logics that can adjust the handles in the first design if the workload is low for the next 2 clock cycles. Third, the “gated memory (2)” design has control logics that can clock/power gate the data memory in addition to adjusting the handles in the second design if the workload is low for the next 2 clock cycles. Fourth, the “gated memory (3)” design has control logics that can adjust the handles in the third design if the workload is low for the next 3 clock cycles.
In all four cases, for <75% zeros, the four designs save 3-7% total energy from small savings in power gating the datapath when loading/unloading the data memory. The small energy savings for highly-non-zero data is likely due to the fact that the low-power techniques such as clock gating, power gating, memory read and write disabling, and voltage scaling were not utilized often. For these cases, the overhead energy of the pre-processing controller still offsets the energy savings from disabling the butterfly datapaths during the FSM’s load and unload stages. Savings become significant for >75% zeros. The gated datapath designs save ~7% total energy for 90% zeros. The significant energy savings for highly-zero data is likely due to the fact that the low-power techniques are utilized most of the time. As most of the data is zero, memory access and leakage is limited. The effect of zeros at the input is propagated down the chains of butterflies. If the majority of data is zero to begin with, the majority of data will continue to stay zero for most of the butterflies in later stages, thus energy savings will continue after the first stage. If only a minority of data is zero however, the data of the butterflies in later stages are unlikely to stay zero after datapath manipulations with non-zero data, thus energy savings are only likely during the first stage.

Figure 5.4, Figure 5.5, Figure 5.6, and Figure 5.7 compare the breakdown of the energy consumed by the data memory, datapath, controller, and ROM/others, respectively for the baseline and the four modified designs. These energy breakdowns are again extracted from post-layout and parasitic
extraction Nanosim simulations. In all four designs, greater energy savings is achieved with more input zeros since the system blocks sleep more often. Clock/power gating the data memory in addition to the datapath saves 3 times the energy as gating the datapath alone. This is consistent with Figure 5.3, which shows that the data memory consumes twice the energy (~50% total) of the datapath (~25%) per FFT. The gated memory designs save 5-40% of total energy in reducing clock transitions and leakage energy in the data memory. Much datapath energy is saved between the 2-cycle and 1-cycle look-ahead designs (5-10% vs. 1-10%), but very little is saved between 3-cycle and 2-cycle. The 2-cycle look-ahead saves more energy than 1-cycle look-ahead due to its lower overhead and since the system is sleeping often. The 3-cycle look-ahead variation saves less energy than 2-cycle due to overhead for small marginal savings. Thus, the optimal design is found to be the 2-cycle look-ahead gated memory and datapath through Nanosim simulations.

**Figure 5.3: Energy Breakdown of Blocks in Serial FFT Systems (0zero)**

![Energy Breakdown Diagram](image)
Figure 5.4: Data Memory Energy

Figure 5.5: Datapath Energy
Figure 5.6: Overhead Energy

Figure 5.7: ROM/Other Energy
The layout of the gated memory system overlaid on the Encounter floorplan is shown in Figure 5.8, with different color blocks indicating different power domains.

Figure 5.8: Modified Serial FFT Layout and Floorplan

5.3 Parallel vs. Modified Parallel vs. Baseline

This section compares the post layout (with standard cell blocks) and post parasitic extraction Nanosim simulation results between the baseline (sFFT), the parallel (pFFT), and the modified parallel (mFFT) design for computing a
32-point FFT. The energy comparisons are based on the test cases with the corresponding clock frequency and minimum supply voltage summarized in Table 5.3.

**Table 5.3: Test Cases and Parameters**

<table>
<thead>
<tr>
<th>FFT Completion Time</th>
<th>Parameters</th>
<th>sFFT (214 cycles)</th>
<th>pFFT (88 cycles)</th>
<th>mFFT (9 cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4280ns</td>
<td>Clock Period</td>
<td>20ns</td>
<td>48ns</td>
<td>475ns</td>
</tr>
<tr>
<td></td>
<td>(Frequency)</td>
<td>(50MHz)</td>
<td>(20.8MHz)</td>
<td>(2.1MHz)</td>
</tr>
<tr>
<td></td>
<td>Min Supply</td>
<td>0.6V</td>
<td>0.5V</td>
<td>0.4V</td>
</tr>
<tr>
<td>2140ns</td>
<td>Clock Period</td>
<td>10ns</td>
<td>24.3ns</td>
<td>237ns</td>
</tr>
<tr>
<td></td>
<td>(Frequency)</td>
<td>(100MHz)</td>
<td>(41.2MHz)</td>
<td>(4.2MHz)</td>
</tr>
<tr>
<td></td>
<td>Min Supply</td>
<td>0.7V</td>
<td>0.5V</td>
<td>0.4V</td>
</tr>
<tr>
<td>1760ns</td>
<td>Clock Period</td>
<td>8ns</td>
<td>19.4ns</td>
<td>142ns</td>
</tr>
<tr>
<td></td>
<td>(Frequency)</td>
<td>(125MHz)</td>
<td>(51.5MHz)</td>
<td>(7.0MHz)</td>
</tr>
<tr>
<td></td>
<td>Min Supply</td>
<td>0.8V</td>
<td>0.5V</td>
<td>0.4V</td>
</tr>
</tbody>
</table>

Figure 5.9 and Figure 5.10 below show the simulation results of the energy per FFT for the two parallel designs compared to the baseline serial design, given various proportions of input zeros. Figure 5.9 is a point-by-point normalization of the new designs to the baseline design for each performance constraint, whereas Figure 5.2 is a normalization of the new designs to the highest energy point of the baseline design (1760ns, 0zeros). Different arrangements of zeros within the inputs, including clumps of zeros at the beginning of an FFT input stream, clumps of zeros at the end, and random zeros throughout, are analyzed. They all give very similar energy results, with their average shown in the graphs below.
From Figure 5.9 for an FFT completion time of 4280ns, the parallel design (pFFT) uses more energy than the baseline design (sFFT) if less than 90% of data are zeros. The high power consumed by the parallel design is likely due to the extra number of flip flops used to store the intermediate multiplier results for odd registers. This is about 1.5x more than the flip flops used in the baseline design, as the baseline design only needed to store the final results from the datapath. A trade-off exists between the energy savings from lower supply voltage, and the energy cost from needing extra flip flops for the parallel design. For other higher performance specs, the pFFT design is always more energy efficient than the sFFT, regardless of the composition of inputs. From Table
5.3, there is a small supply voltage difference of 0.1V between the parallel
design and the baseline design for FFT completion time of 4280ns. However, a
larger supply voltage difference of 0.2V and 0.3V exists for the lower
performance spec of 2140ns and 1760ns, respectively, likely resulting in the
more fruitful energy savings observed. For all simulated completion time, the
modified parallel design (mFFT) uses an order of magnitude lower energy
compared to the baseline design, regardless of the inputs. Such a great
reduction in energy consumption is likely due to the fact that the supply voltage
can be lowered to near-threshold voltage, since the low cycle count allow for a
much slower clock. The modified parallel design also does not need the extra
flip-flops that the parallel design needed, as only the final datapath results are
stored.
Figure 5.10: Normalized (Highest-Point) Energy Simulation Results vs. % of zeros

Figure 5.11 and Figure 5.12 below show the simulation results of the energy per FFT for the two parallel designs compared to the baseline serial design, given various FFT completion time constraints. Figure 5.9 is a point-by-point normalization of the new designs to the baseline design, whereas Figure 5.2 is a normalization of the new designs to the highest energy point of the baseline design.
Figure 5.11: Normalized (Point-by-Point) Energy Simulation Results vs. Performance
Figure 5.12: Normalized (Highest-Point) Energy Simulation Results vs. Performance

From Figure 5.11, for an idle FFT (when all inputs are zero), the pFFT design is always more energy efficient than the sFFT design, regardless of performance constraint. For this type of input data, the energy expenditures from the extra flip flops are low, so the energy savings from lower supply voltage dominates. For inputs with other proportions of zeros, the trade-off between energy savings from lower supply voltage and energy cost from extra flip flops are shown through the figure. There are corresponding performance specifications up to which it makes sense to use pFFT instead of sFFT. For all combinations of performance spec and data composition, the mFFT design is always a winner in terms of being the most energy efficient.
In summary, from the simulation results, the parallel FFT is better or worse than the baseline design for certain performance requirements, but the modified parallel FFT is always more energy efficient. This is however, at the expense of requiring a larger area. The layouts of the parallel and modified parallel designs overlaid on their corresponding Encounter floorplans are shown in Figure 5.13 and Figure 5.14, respectively.

Figure 5.13: Parallel FFT Layout and Floorplan
Figure 5.14: Modified Parallel FFT Layout and Floorplan
Chapter 6

Conclusions

This thesis explored several new 32-point radix-2 FFT architectures that take advantage of data dependency and parallelism to save energy. In summary, the modified serial architecture and the modified parallel architecture can reduce the energy per FFT by up to 45% and 90%, respectively, when compared to the baseline design.

For the modified serial design, the ideas of dynamically optimizing energy within each FFT (as opposed to between FFT computations), regulating a large data memory with a small representative control memory, and designing a controller that looks ahead at a sequence of input data to determine the best optimization for the next iteration are pursued. The overall contribution is a fully working low-energy pre-processing controller not in the critical computation path that can predict the workloads of future iterations and adjust
the performance of the critical path accordingly. A low-voltage control memory is incorporated that stores one bit for each 16-bit word in the data memory indicating whether it is zero. The control logic determines the workload required in each iteration based on how many inputs are zero, and then disables/voltage-scales the data memory, disable/power gates the datapath, or disables the ROM. Bits from the control memory are pipelined to the control logic so it can consider the workload of future iterations. Through Nanosim simulations of four variation of the design, the 2-cycle look-ahead gated memory and datapath design appeared to be the most optimal.

For the parallel and modified parallel design, the idea of basing the parallel architecture on a different flow graph, where each stage is completely identical is pursued. This can eliminate the need for routing large amounts of data dynamically for each new butterfly stage. The overall contribution is a fully working parallel architecture. The datapath of the modified parallel design is simply multiple instantiations of the datapath from the baseline design. Through Nanosim simulations, it is demonstrated that the single-cycle arithmetic unit is indeed better than the multi-cycle heavily pipelined arithmetic unit. This is because the fewer cycle count allows for the clock frequency to be greatly reduced, which in turn allow for a reduction in supply voltage.

Note that due to the lack of readily available SRAM in gpdk090, flip-flop based memories are used instead. Future work could include incorporating the
SRAM into the modified serial FFT design to see its impact on energy savings. For small storage capacity, such as the case for 32-point FFT, the overhead of SRAM's peripheral circuitry such as the precharge circuitry and sense amplifier may lead to bigger area and possibly more energy consumption compared to flip-flop based memories [14]. However, similar relative energy savings are still expected from the modified serial design when compared to the baseline design. It is also recommended that further exploration is done on different parallel implementations. Particularly, as recommended by Professor Chandrakasan, a fully expanded parallel architecture, where all FFT butterflies are performed simultaneously (as opposed to only the ones in the same stage) could be developed. In this scheme, dedicated hard-wired multipliers and adders could be developed for different twiddle factors. This is in contrast with the generic multipliers used in all the datapaths within this thesis. Lastly, only the energy savings for N=32 (32-point FFT) were explored here. It would be interesting to see if such energy savings continue to scale with higher-point FFT (e.g. 1024-point FFT). For higher-point FFT, bigger relative energy saving is expected from the modified serial design as the energy savings from the data memory and datapath becomes more dominant over the cost of the controller overhead energy. For the modified parallel design, greater relative energy saving is also expected, as the cycle count ratio between the modified parallel design and baseline design, \( \frac{N}{2} \log_2 N \), continues to diminish. The bigger frequency differences allow more differences between the supply voltages.
Appendix A

Alternative Forms of FFT Flow Diagrams

Figure A.1: FFT Decimation-in-Time with Input in Normal Order [1]
Figure A.2: FFT Decimation-in-Time with Input and Output in Normal Order [1]

Figure A.3: FFT Decimation-in-Frequency with Input in Normal Order [1]
Appendix B

Baseline Implementation Details

Table B.1: Input Ordering vs. Address Locations of an 8-point FFT

<table>
<thead>
<tr>
<th>FFT inputs $x[n]$</th>
<th>$N$ in binary</th>
<th>address $k$ (bit-reversed $n$)</th>
<th>FFT outputs $X[k]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x[0]$</td>
<td>000</td>
<td>000</td>
<td>$X[0]$</td>
</tr>
<tr>
<td>$x[1]$</td>
<td>100</td>
<td>010</td>
<td>$X[1]$</td>
</tr>
</tbody>
</table>
Table B.2: Address Generation for an 8-point FFT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>$N$-point FFT</td>
</tr>
<tr>
<td>$n = \log_2 N$</td>
<td># of stages, also # of bits in addr</td>
</tr>
<tr>
<td>$b = \frac{N}{2}$</td>
<td># of butterflies per stage</td>
</tr>
<tr>
<td>$i = 0, 1, ..., n-1$</td>
<td>$i^{th}$ stage</td>
</tr>
<tr>
<td>$j = 0, 1, ..., b-1$</td>
<td>$j^{th}$ butterfly in $i^{th}$ stage</td>
</tr>
<tr>
<td>$m = {j[n-2:1], 0}$</td>
<td>intermediate value,</td>
</tr>
<tr>
<td>$u = {j[0], ROL_{n-1}(m, i)}$</td>
<td>A's address in memory</td>
</tr>
<tr>
<td>$v = {j[0], ROL_{n-1}(m+1, i)}$</td>
<td>B's address in memory</td>
</tr>
<tr>
<td>$k = j$ with $(n-1-i)$LSB set to 0</td>
<td>W's address in ROM</td>
</tr>
</tbody>
</table>

*ROL_{n-1}(a, \beta)$ is a function that produces $n-1$ bits by taking the value of $a$ and ring rotate the bits to the left by $\beta$.

Table B.3: Address Generation for an 8-point FFT

<table>
<thead>
<tr>
<th>$N = 8$</th>
<th>$n = 3$</th>
<th>$b = 4$</th>
<th>$i = 0, 1, 2$</th>
<th>$j = 0, 1, 2, 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>$j$</td>
<td>$m$</td>
<td>$u$</td>
<td>$v$</td>
</tr>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>000</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>00</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>010</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>10</td>
<td>110</td>
<td>111</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>00</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>00</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>001</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>10</td>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td>000</td>
<td>100</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>001</td>
<td>101</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>011</td>
<td>111</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>010</td>
<td>110</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Appendix C

Booth Multiplier Algorithms

Table C.1: Booth Algorithm LSB Combinations [11]

<table>
<thead>
<tr>
<th>LSB</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1. Do nothing&lt;br&gt;2. Shift product to the right (sign extend MSB)</td>
</tr>
<tr>
<td>01</td>
<td>1. Add product by multiplicand (ignore any overflow)&lt;br&gt;2. Shift product to the right (sign extend MSB)</td>
</tr>
<tr>
<td>10</td>
<td>1. Subtract product by multiplicand (ignore any overflow)&lt;br&gt;2. Shift product to the right (sign extend MSB)</td>
</tr>
<tr>
<td>11</td>
<td>1. Do nothing&lt;br&gt;2. Shift product to the right (sign extend MSB)</td>
</tr>
</tbody>
</table>
Table C.2: Booth Algorithm Example 1

**Example 1:** $m \times r = -8 \times 7 = -56$

Multiplicand and multiplier are each 4 bits

<table>
<thead>
<tr>
<th>Variable Setup</th>
<th>Variable Setup</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m = (-8)_{10} = (11000)_2$</td>
<td>$-m = (+8)_{10} = (01000)_2$</td>
</tr>
<tr>
<td>$r = (+7)_{10} = (0111)_2$</td>
<td>$S = -m, 0000 = 01000000$</td>
</tr>
<tr>
<td>$A = {m, 0000} = 110000000$</td>
<td>$P = 00000, r, 0} = 000001110$</td>
</tr>
</tbody>
</table>

Perform Calculation Loop:

<table>
<thead>
<tr>
<th>Clock cycle</th>
<th>$P$</th>
<th>Actions</th>
</tr>
</thead>
</table>
| 0 | 0000001110 | 1. $P = P + S \rightarrow 0100001110$
2. $P \gg 1 \rightarrow 0010001110$ |
| 1 | 0010001111 | 1. $P \gg 1 \rightarrow 0001000111$ |
| 2 | 0001000111 | 1. $P \gg 1 \rightarrow 00001000001$ |
| 3 | 00001000001 | 1. $P = P + A \rightarrow 110010000001$
2. $P \gg 1 \rightarrow 1110010000$ |
| 4 | 1110010000 | |

$(11001000)_2 = (-128 + 64 + 8)_{10} = (-56)_{10} \rightarrow \text{correct}$
Bibliography


