The Dissertation Committee for Bassam Jamil Mohd
certifies that this is the approved version of the following dissertation:

Switch-Based Fast Fourier Transform Processor

Committee:

Earl E. Swartzlander, Jr., Supervisor

Mircea Driga

Mohamed Gouda

Michael Orshansky

Nur Touba

Switch-Based Fast Fourier Transform Processor

by

Bassam Jamil Mohd, B.S.; M.S.

Dissertation

Presented to the Faculty of the Graduate School of

the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December, 2008

In the Name of My Lord,

To my parents, my wife and my children.

Acknowledgements

First, I want to thank the Lord for providing me the will, the time and the energy to accomplish this work. I am grateful to my parents, my wife and my children for their love and support. I greatly appreciate their prayers throughout this endeavor.

I would like to thank my advisor, Professor Earl E. Swartzlander, for his guidance, understanding and support throughout the course of this research. Working with him has been a great experience and great fun. Thanks to my committee members for their ideas and invaluable feedback. I would like also to thank Professor Adnan Aziz for his guidance and support. Thanks to the Electrical and Computer Engineering staff for their assistance.

I would like to express my gratitude to my friends for their encouragement and assistance. Special thanks to Hani Saleh, Adel Husain, Mohammed Baker, Adnan Suleiman and Mohammed Zeidan. Also, special thanks to my proofreader Tim Heierman for great feedback, as well as, special thanks to Asad Bawa for his assistance with the backend tool. Also, special thanks to my colleagues and managers at my current and previous employer for their understanding and encouragement.

Switch-Based Fast Fourier Transform Processor

Publication No. _____________________

Bassam Jamil Mohd, Ph.D.
The University of Texas at Austin, 2008

Supervisor: Earl E. Swartzlander, Jr.

The demand for high-performance and power scalable DSP processors for telecommunication and portable devices has increased significantly in recent years. The Fast Fourier Transform (FFT) computation is essential to such designs. This work presents a switch-based architecture to design radix-2 FFT processors. The processor employs M processing elements, $2M$ memory arrays and M Read Only Memories (ROMs). One processing element performs one radix-2 butterfly operation. The memory arrays are designed as single-port memory, where each has a size of $N/(2M)$; N is the number of FFT points. Compared with a single processing element, this approach provides a speedup of M. If not addressed, memory collisions degrade the processor performance. A novel algorithm to detect and resolve the collisions is presented. When a collision is detected, a memory management operation is executed. The performance of the switch architecture can be further enhanced by pipelining the design, where each pipeline stage employs a switch component. The result is a speedup of $M\log_2N$ compared with a single processing element performance.

The utilization of single-port memory reduces the design complexities and area. Furthermore, memory arrays significantly reduce power compared with the delay elements used in some FFT processors. The switch-based architecture facilitates deactivating processing elements for power scalability. It also facilitates implementing different FFT sizes.

The VLSI implementation of a non-pipeline switch-based processor is presented. Matlab simulations are conducted to analyze the performance. The timing, power and area results from RTL, synthesis and layout simulations are discussed and compared with other processors.

Table of Contents

Acknowledgements v

Switch-Based Fast Fourier Transform Processor vi

Table of Contents viii

List of Figures x

List of Tables xii

Chapter 1 Introduction 1

Chapter 2 Background 4

- 2.1 Radix-2 FFT Algorithm 6
- 2.2 Bass: The Cached-FFT Processor 8
- 2.3 Pipeline FFT 12
- 2.4 El-Khashab, *et al.*: the Modular Pipeline FFT 14
- 2.5 Zhong, *et al.*: the Processor Ring Architecture 21
- 2.6 Cohen: Address Generation Scheme 24
- 2.7 Ma: Reducing the Complexity and Power of Address Generation 27
- 2.8 Summary 32

Chapter 3 Research Methodology: Approach and Flow 34

- 3.1 Research Approach 34
- 3.2 Design Flow 35
- 3.3 Summary 39

Chapter 4 Architecture and Algorithm 40

- 4.1 Switch-Based Architecture 40
  - 4.1.1. Non-pipeline Switch-Based [22] 40
  - 4.1.2. Pipeline approach [23] 45
  - 4.1.3. Comparing the pipeline and non-pipeline approaches 48
- 4.2 Memory Management Algorithm 49
  - 4.2.1. Algorithm Pseudocode 54
- 4.3 64-point Design 57
- 4.4 Dynamic Range and Signal to Noise Ratio 63
- 4.5 Summary 65

Chapter 5 Logic Design 66

- 5.1 Design Specifications 66
- 5.2 Design Overview 67
- 5.3 Functional Block Descriptions 69
  - 5.3.1. Memory Sub-System 70
  - 5.3.2. The Switch Sub-System 75

5.4 Configuration and Operation 93
5.5 Hardware and Timing Complexities 96
5.6 Summary 97
Chapter 6 RTL Simulation and Synthesis 98
6.1 RTL Simulation Results 98
6.2 RTL Synthesis 105
    6.2.1. Timing Analysis 105
6.3 Power Estimation 110
6.4 Summary 115
Chapter 7 Physical Design 117
7.1 Physical Design 117
7.2 Timing Analysis 122
7.3 Power Analysis 124
7.4 The Impact of Wire Resistance and Capacitance 128
7.5 Comparison with other Processors 129
7.6 Summary 132
Chapter 8 Conclusion 134
8.1 The Key Contributions 134
8.2 Future Research 135
Appendix: Wallace Multiplier 136
Bibliography 138
VITA 142

List of Figures

Figure 1. FFT Processor Architectures [11] 5
Figure 2 Radix-2 Butterflies [10] 7
Figure 3. 16-point radix-2 DIF FFT 7
Figure 4 Dataflow for a Single Memory 64-point DIT FFT [10] 9
Figure 5. Dataflow for a Cached FFT 64-point DIT [10] 11
Figure 6. SDF and MDC Radix-r FFT Pipelines 13
Figure 7. Basic Flow of the Modular Pipeline FFT Using a 64-Point Example 15
Figure 8. 16-Point FFT Butterfly Diagram with Intermediate Values 17
Figure 9. 16-Point FFT Butterfly with Identical First and Second Modules 17
Figure 10. Radix-2 Modular Pipeline Architecture 18
Figure 11. Input Address Generation Logic 19
Figure 12. Output Address Generation Logic 20
Figure 13. Processor Ring 22
Figure 14. Address Generation Logic [17] 26
Figure 15. Coefficients in 16-Point DIT FFT Calculations 28
Figure 16. Read Address Generation 31
Figure 17. Write Address Generation 31
Figure 18. Research Approach 35
Figure 19. Design Flow 36
Figure 20. Power Estimation Flow [24] 39
Figure 21. Switch-Based Architecture 41
Figure 22. Switch Block Diagram 42
Figure 23. PE Block Diagram 44
Figure 24. 16-Point DIF FFT 45
Figure 25. Switch-Based Pipeline 47
Figure 26. Memory Management Operations 51
Figure 27. Tracking Shuffled Intermediate Results Using Pointers 52
Figure 28. Memory Conflict Free Algorithm 53
Figure 29. Memory Conflict Free 16-Point FFT 54
Figure 30. Design Overview 68
Figure 31. Block Processing Timing Diagram 69
Figure 32. External Memory Controller State machine 73
Figure 33. Routing address and data in the design 74
Figure 34. The Three Portions of External Memory Address 75
Figure 35. Switch Connectivity with the Design Components 77
Figure 36. Switch State Machine 78
Figure 37. Blocks, Stages, Cycles and Clock Relationship 79
Figure 38. ROM Address Generation 80
Figure 39. Verilog Code for ROM Address Generation 81
Figure 40. RAM Address Generation 81
Figure 41. Verilog Code for RAM Address Generation 82
Figure 42. Verilog Code for PE MUX Select Generation 83
Figure 43. Verilog Code for RAM Mux Select Generation 83
Figure 44. PE Block Diagram 85
Figure 45. Shuffle and Swap Signal Generation 86

Figure 46. Normalized Power Results for Disabling the MUL 87
Figure 47. Breakdown of PE Power Across Experiments 87
Figure 48. Generating 16-bit Fractional Format from the MUL output 88
Figure 49. Routing ROM Data to PE_{pe_id} 90
Figure 50. ROM Access Timing Path 91
Figure 51. Reducing ROM Power using Address-Gating 92
Figure 52. Processor Configuration Timing Diagram 94
Figure 53. External Memory Read Access 95
Figure 54. External Memory Write Access 95
Figure 55. Test signal x_a and x_b 99
Figure 56. 64-point RTL-Generated FFT 100
Figure 57. 256-point RTL-Generated FFT 101
Figure 58. 1024-point RTL-Generated FFT 102
Figure 59. 4096-point RTL-Generated FFT 103
Figure 60. PE Timing Path 106
Figure 61. External Memory Controller Timing Path 109
Figure 62. Reporting Cell Power 111
Figure 63. Average power vs. Number of Active PEs 114
Figure 64. Component Power Across Tests 115
Figure 65. Component Power Across Tests 115
Figure 66. Floorplan 118
Figure 67. Layout View 121
Figure 68. PE Timing Path 123
Figure 69. Power vs. Number of Active PEs 127
Figure 70. Component Power vs. Number Active PEs 128
Figure 71. Impact of RC on Power and Timing 129
Figure 72. N-bit by N-bit Wallace Multiplier [42] 136
Figure 73. A 6-bit by 6-bit Wallace Multiplier 137

List of Tables

Table 1. Cached FFT Variables [10] 10
Table 2. Butterfly Addresses, Cache Address and W_N Coefficients for a 64-Point, Radix-2, DIT, 2-Epoch Cached FFT [10] 12
Table 3. SDF and MDC Comparison [5] 14
Table 4. Radix-2 Address Sequence for Two 16-Point FFTs 19
Table 5. Complexity of Radix-r Standard and Modular Pipeline FFTs 21
Table 6. Calculating X and Y 23
Table 7. Processor Ring Program Using Two Processors 24
Table 8. Standard Butterfly Sequence for a 16-Point FFT 24
Table 9. Cohen Butterfly Sequence for a 16-Point FFT 25
Table 10. Cohen Address Generation for Stage Two of a 16-Point FFT 25
Table 11. Reordered Butterflies for 16-point FFT 29
Table 12. Counters and Variables Used in Address Generation 30
Table 13. Memory Partition and Data Assignment 30
Table 14. Summary of the Approaches 33
Table 15. Wireload Model Format 38
Table 16. Number and Size of ROMs per Pipeline Stage 46
Table 17. Comparing Switch-Based Pipeline FFT with Other Pipeline FFTs 48
Table 18. Comparing Pipeline and Non-Pipeline Designs 49
Table 19. The Main Variables and Counters Used in the Algorithm 55
Table 20. Stage-0 PE Operand Pairs 58
Table 21. Stage-1 PE Operand Pairs 59
Table 22. Stage-2 PE Operand Pairs 59
Table 23. Stage-3 PE Operand Pairs 59
Table 24. Stage-4 PE Operand Pairs 59
Table 25. Stage-5 PE Operand Pairs 60
Table 26. Memory-0 Contents Across FFT Stages 60
Table 27. Memory-1 Contents Across FFT Stages 60
Table 28. Memory-2 Contents Across FFT Stages 61
Table 29. Memory-3 Contents Across FFT Stages 61
Table 30. Memory-4 Contents Across FFT Stages 61
Table 31. Memory-5 Contents Across FFT Stages 62
Table 32. Memory-6 Contents Across FFT Stages 62
Table 33. Memory-7 Contents Across FFT Stages 62
Table 34. Dynamic Rage for Various Fixed-Point Representations 63
Table 35. SNR for Various Fixed-Point Representations 65
Table 36. Design Specifications 67
Table 37. RAM Interface 71
Table 38. RAM Timing Specification 71
Table 39. Energy/Power Specification 71
Table 40. Register Specifications 92
Table 41. Configuration Vector 93
Table 42. Hardware Complexity 96
Table 43. Number of Clocks for Different FFT Configurations 97
Table 44. Noise Floor for Various FFT Plots 104

Table 45. PE Timing Path 107
Table 46. ROM Access Timing Path 108
Table 47. External Memory Address Generation Timing Path 110
Table 48. Post Synthesis Power Results 112
Table 49. Post Synthesis Results Summary 116
Table 50. Cell Statistics 120
Table 51. Wire Statistics 120
Table 52. Nets and Pins Statistics 122
Table 53. Timing Delays in the PE Timing Path 124
Table 54. Delay Percentages for the PE Timing Path 124
Table 55. Layout Power Results 125
Table 56. Benchmarks of Fixed-Point Processors for 1024-Point FFT 130
Table 57. Benchmarks of Fixed-Point Processors for 256-Point FFT 130
Table 58. Samples per Second Criterion 131
Table 59. Comparing “Energy x Time” Criterion 132
Table 60. Data Sheet 133

Chapter 1

Introduction

The demand for high performance and energy efficient digital signal processors (DSPs) has significantly increased recently primarily driven by phones and portable devices. The Fast Fourier Transform (FFT), proposed by [1], plays a key role in the DSP design. FFT processor design has been researched extensively in the last few decades. As a result, many implementations have been proposed and developed to address one or more of the following optimization areas: architecture, memory access, and power. However, many of the existing designs do not fully exploit the FFT parallelism due to inefficient topology or suboptimal memory access. Other designs suffer power dissipation issues because of employing delay elements and decomposition techniques. Moreover, most of the designs offer limited configurability and power scalability features. The problem addressed by this dissertation is the limited performance, configurability and power scalability of current FFT processors.

This research presents the switch-based architecture that consists of a switch fabric, M processing elements (PEs) and $2M$ memories. Each processing element implements a radix-2 FFT butterfly operation. The memory elements are single-port and each has a capacity of $N/(2M)$ entries. Furthermore, the twiddle factors are stored in M Read Only Memories. The throughput increases by a factor of M compared to a single PE. The architecture supports pipelined implementation, where each pipeline stage employs a single switch. This further improves the throughput by a factor of the number of stages. The pipelined implementation can reach a speedup of $M*\log_2 N$ compared to a single PE.

If left unresolved, memory contentions degrade the design performance. A memory contention occurs when multiple requests are made to the same physical memory. A memory management algorithm was developed to detect and resolve contentions. Once a potential hazard is detected, the algorithm executes memory management operations to swap on-the-fly data being read from or written to the memory. Additionally, with minimal overhead, the switch architecture can support multiple sizes of FFT. Further, by configuring the number of active processing elements, the processor power is scaled based on the system power/performance requirements.

The goal of this research is to develop and implement the architecture, algorithm, logic and physical design for an energy-efficient high-performance radix-2 FFT switch-based processor. The research is intended to achieve the following sub-goals:

1. Develop a switch-based architecture for non-pipeline and pipeline processors.
2. Develop an algorithm to detect and resolve memory contention for M PEs.
3. Design and implement a non-pipeline switch-based processor. The processor employs up to four PEs and is configurable to process 64-, 256-, 1024- and 4096-point FFTs. A configuration interface is implemented to select the number of active PEs.
4. Develop and implement the processor RTL and physical design. The physical design is implemented using a low power 65 nm CMOS process.
5. Quantify the performance (area, delay, and power consumption) of the physical design. Moreover, the switch overhead in terms of timing and power dissipation is analyzed.

When compared to other FFT designs, the primary benefits of the switch processor are:

1. A significant speed increase (up to M for non-pipelined and $M\log_2 N$ for pipelined) over a single processing element.
2. The elimination of performance degrading memory contentions.
3. The utilization of single-port memories which reduces the design complexity and area.
4. The use of RAMs for data storage, which significantly reduces power compared with the delay elements (i.e., shift registers) used by other FFT designs.

5. The ability scale the power linearly based on the number of active processing elements.
6. The ability to configure the processor to accommodate different FFT sizes.

The remaining chapters of the dissertation are organized as follows. First, several FFT architectures and implementations are investigated in detail. Next, the general dissertation approach is illustrated. The architecture and algorithm are then presented, followed by a detailed discussion of the RTL implementation. The synthesis and layout simulations and results are discussed in detail. The final chapter presents concluding remarks and future research ideas.

Chapter 2

Background

Many FFT processor designs have been proposed and implemented to address one or more of the following optimization areas: architecture, memory access and power.

A variety of FFT architectures have been proposed, which employ different techniques such as caches, multi-processing and pipelining, as shown in Figure 1 [11]. The *single memory* architecture consists of a processor connected to a single N-word memory via a bidirectional bus. While this architecture is simple, its performance suffers from inefficient memory bandwidth. The *cache memory* architecture adds a cache memory between the processor and the memory to increase the effective memory bandwidth. The *dual memory* architecture uses two memories connected to a single processor. A memory controller generates addresses to memories in a ping-pong fashion. The *pipeline FFT* architecture utilizes multiple pipeline stages; each pipeline stage consists of delay lines and butterfly processors. Lastly, the *processor array* architecture consists of an array of independent processing elements, utilizing local buffers, which are connected using an interconnect network.

The processor memory access is another area of optimization which has received considerable research. Several algorithms have been proposed to resolve memory contention. In particular, the address generation algorithm and logic have been optimized for speed and area.

Further, several power reduction techniques have been designed for energy-efficient processors; including techniques to reduce the number of memory accesses. Because of the dynamic environment of current designs, it is essential to manage and scale the processor power. As a result, power scalability and configurability have received considerable attention as well. Additionally, configuring the FFT processor to compute different FFT sizes is highly desirable.

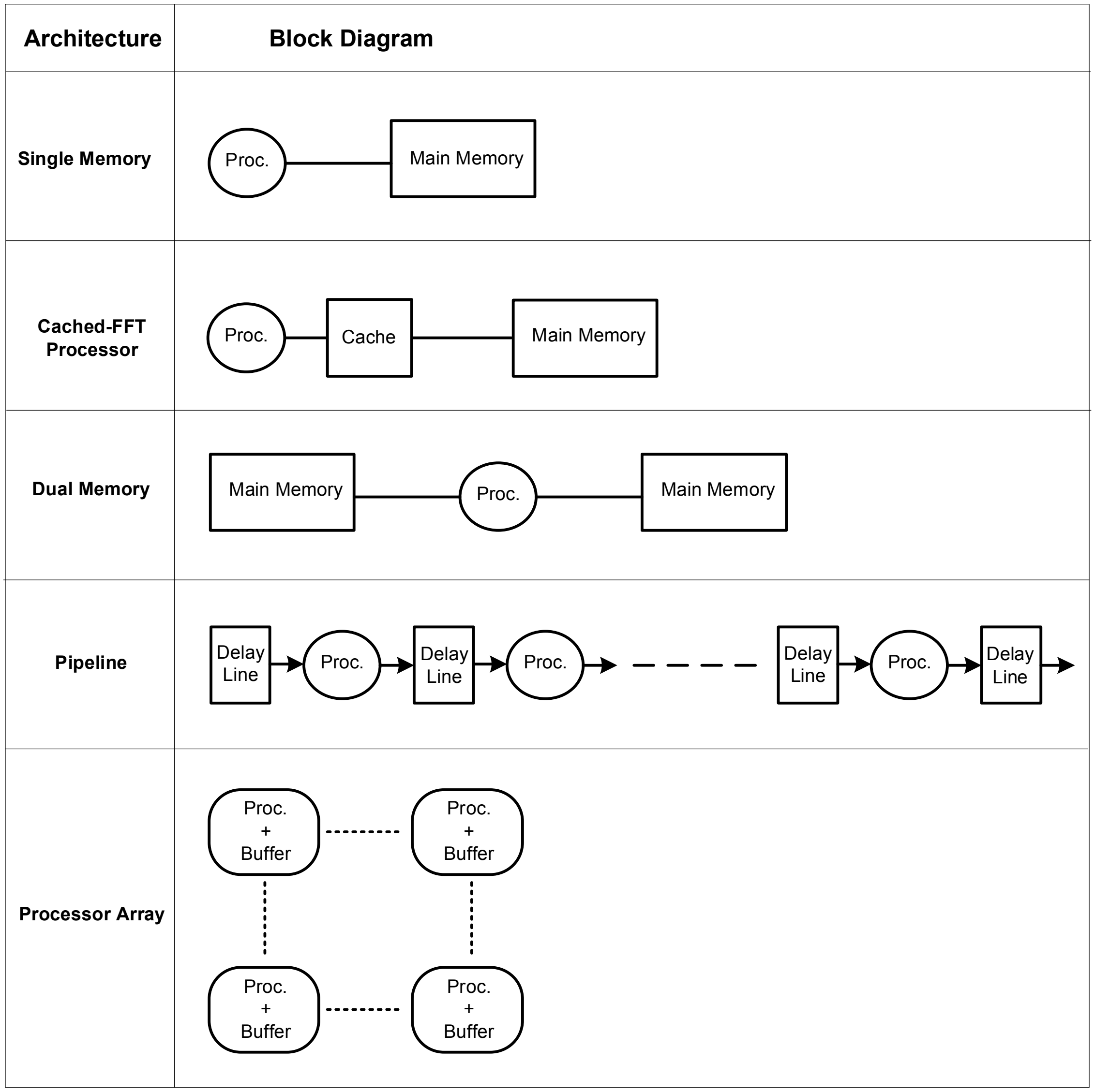


Figure 1. FFT Processor Architectures [11]

The word "stage" is overused in the FFT literature. It is used to refer to one stage (i.e., one pass) of the FFT algorithm (e.g., [28]). It is also used to describe to one hardware stage (i.e., module) in the pipeline FFT design (e.g., [3]). To avoid any confusion, the dissertation will follow this convention. The hardware pipeline stage is referred to as "pipeline stage." The FFT algorithm stage is referred to as stage (with no qualification).

The remainder of this chapter starts with a brief description of the radix-2 FFT algorithm. Next, a survey of key research and contributions in the above optimization areas is presented.

2.1 Radix-2 FFT Algorithm

Fast Fourier Transform (FFT) refers to the set of highly efficient computational algorithms which calculates N-point DFT [29]. In 1965 James Cooley and John Tukey published one of the earliest FFT algorithms [1]. Since then many FFT algorithms, which employ different radices or decomposition techniques, have been proposed and published. These algorithms are well explained in many textbooks and publications, e.g., [9], [14], [29]. Because it is used in this research, the remainder of this section focuses on the decimation-in-frequency radix-2 FFT algorithm.

The radix-2 algorithm consists of $\log_2 N$ stages. Each stage receives N input points and computes N output points. The basic computation performed in each stage is the butterfly operation, shown in Figure 2. The butterfly receives two input points and a twiddle factor. It performs complex addition and multiplication operations, as illustrated in Figure 2, and produces two outputs. Depending on the type of the butterfly and the order of input points, there are two equivalent types of radix-2 FFT: decimation-in-time (DIT) and decimation-in-frequency (DIF). The following discussion focuses on the DIF radix-2 FFT.

Figure 3 shows a 16-point radix-2 DIF FFT example. There are $N/2$ butterfly operations performed on a pair of data in each stage. Figure 3 shows eight butterfly operations in each stage. In stage 0, x(0) and x(8) are paired. The spacing for the pairs in stage 0 is eight. In stage 1, the spacing is four. In general, for stage i, x(j) and x(k) are paired in one butterfly operation, if:

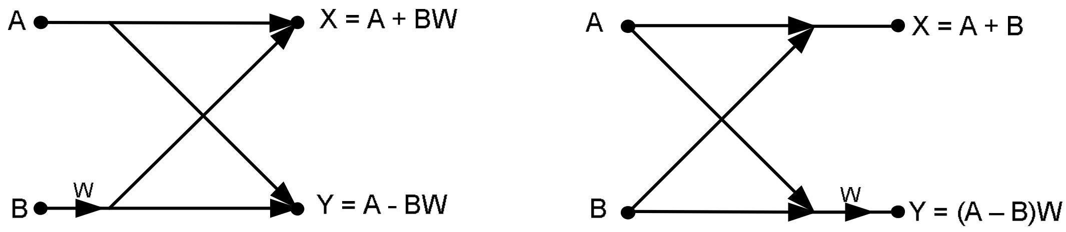
$k = j + 2^{(\text{Number\_of\_Stages} - i)}$, where,

$\text{Number\_of\_stages} = \log_2 N$

The operations in stage i are divided into 2^i groups, where each group (g) has $N/2^{(i+1)}$ butterfly operations. For example, in Figure 3, stage 1 has two groups, where each group has four butterfly operations. The twiddle factor for butterfly (b) in group (g) of stage i is equal to $W_N^{b\times 2^i}$, where:

$$W_N = e^{-i2\pi/N} = \cos\left(\frac{2\pi}{N}\right) - i\sin\left(\frac{2\pi}{N}\right)$$

For example, in stage 1, the twiddle factors are $W_{16}^0, W_{16}^2, W_{16}^4$ and W_{16}^6 in the first and second groups.



Raidx-2 Decimation In Time Butterfly

Radix-2 Decimation In Frequency Butterfly

Figure 2 Radix-2 Butterflies [10]

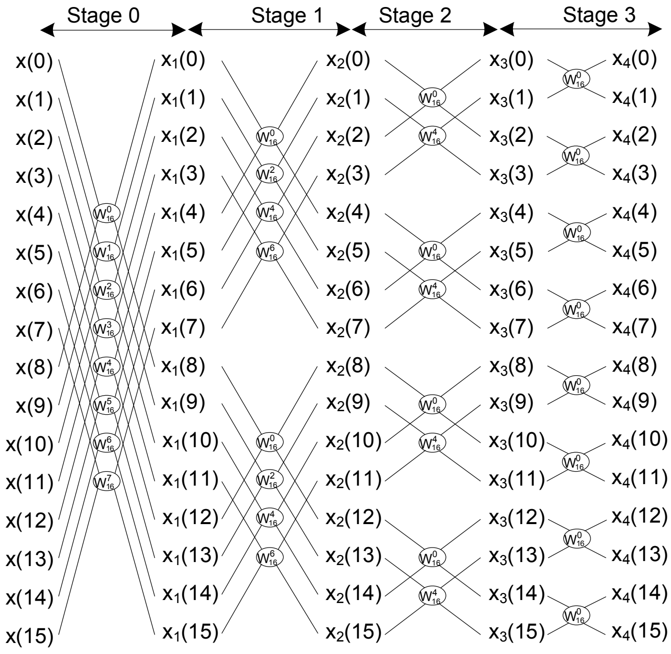


Figure 3. 16-point radix-2 DIF FFT

For this version of the FFT, the inputs of stage 0 are ordered normally. However, the output of the last stage requires reordering. The reordering is done by bit-reversal of the output index. Overall, the complexity of the radix-2 FFT algorithm is $O(N\log_2 N)$. This is a substantial improvement compared to Discrete Fourier Transform, which has a complexity of $O(N^2)$ [29].

2.2 Bass: The Cached-FFT Processor

Baas introduced a cached-based architecture in [10]-[14], which increases the processor frequency and reduces the energy dissipation compared to a single memory architecture. By adding a cache unit between the processor and memory, the processor communicates data with the cache instead of memory, which enables higher operating clock frequencies. Further, compared to memory, the processor cache has less energy access cost (i.e., Joules/access); this results in reducing data communication energy. The main disadvantages of this approach are the addition of a new functional unit (i.e., the cache) and the added cache controller complexity.

Baas made the following observations about executing a standard radix-2 FFT using a single-memory architecture (shown in Figure 1). A 64-pioint standard radix-2 FFT is illustrated in Figure 4. First, the processor speed is severely limited by the memory bandwidth and is tightly coupled with memory access speed. In addition, FFT input and output data have to be transported from/to memory to/from the CPU for each butterfly operation. This requires accessing larger memory arrays and toggling longer data wires, and therefore, this dissipates higher energy. Baas proposed dividing the FFT computations into epochs, groups and passes as demonstrated in Figure 5. An epoch is the portion of the cached-FFT algorithm where all N data words are loaded in to a cache, processed, and written back to main memory once. A group is the portion of an epoch where a block of data is read from the main memory into a cache, processed, and written back to main memory. A pass (i.e., FFT stage) is the portion of a group where each word in the cache is read, processed with a butterfly, and written back to the cache once. The basic idea of the cached-FFT algorithm is to decompose N-point FFT computation using C-point epochs.

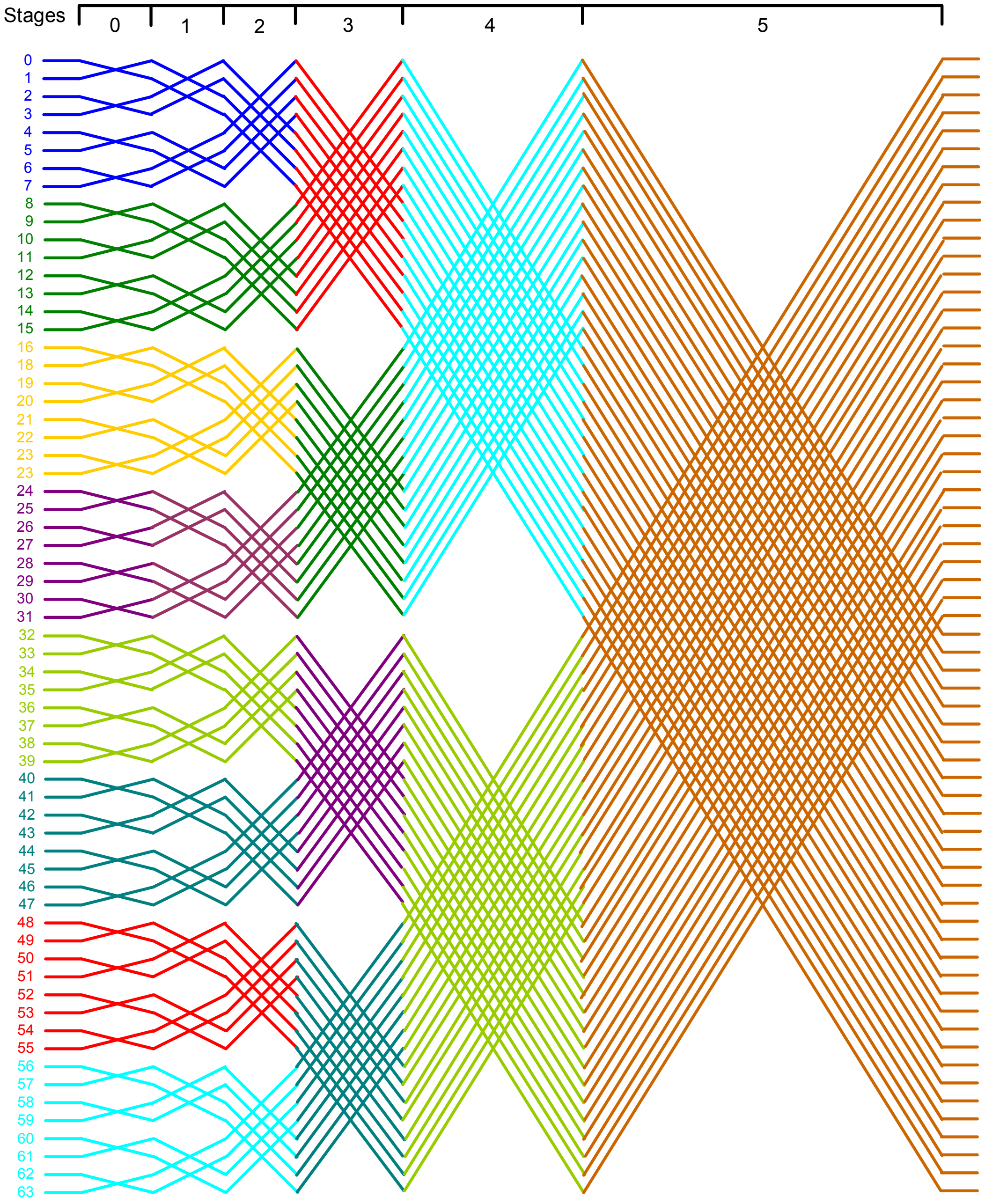


Figure 4 Dataflow for a Single Memory 64-point DIT FFT [10]

For each epoch, the groups are loaded one by one in the cache. Once loaded into the cache, the group points will be processed for a C-point FFT. Finally, the group is written back to memory and the next group is loaded. Epochs are processed serially because of data dependency. Shown in Figure 5, there are two epochs, which correspond to e = 0 and e = 1; each epoch has eight 8-point FFTs.

The following pseudo-code algorithm summarizes the cached-FFT algorithm:

```
• For each epoch e where e=0,1, … (E-1)
    • For each group g in e, where g=0, 1, … (N/C-1)
        • Load group g from memory to cache,
        • Perform C-point FFT on cache data,
        • Store cache results of group g to memory.
```

Baas provided formulas to calculate key design preparers, such as cache size and number of epochs. Table 1 illustrates how those parameters are computed for general cases as well as the 64-point example in Figure 5.

Table 1. Cached FFT Variables [10]

Variable	**Expression**	**Design in Figure 5**
Cache Size	$C = \sqrt[E]{N}$	*C=8*
Number of Epochs	$E = \left\lceil \frac{\log_r N}{\log_r C} \right\rceil$	*E=2*
Number of Groups per Epoch	*N/C*	*8*
Number of Passes per Group	$(\log_r N)/E = \log_r C$	*3*
Number of Butterflies per Pass	*C/r*	*4*

Table 2 shows the addresses and coefficients for a 64-point cached FFT. Note the following:

- $[b_1,b_0]$: 2-bit butterfly counter
- $[g_2,g_1,g_0]$: 3-bit group counter
- “*” stands for 0 or 1

For example, for epoch 0, pass 2:

- The butterfly addresses are $[\ g_2\ \ g_1\, g_0 * b_1\, b_0]$,
- The cache addresses are $[* \ b_1 b_0]$,
- The twiddle factors are $W_{64}^{\,b_1 b_0 000}$.

In summary, the cached FFT approach increases the clock speed and reduces the energy required to move intermediate results from/to main memory. The main disadvantage of this approach is the overhead of the cache unit and its controller.

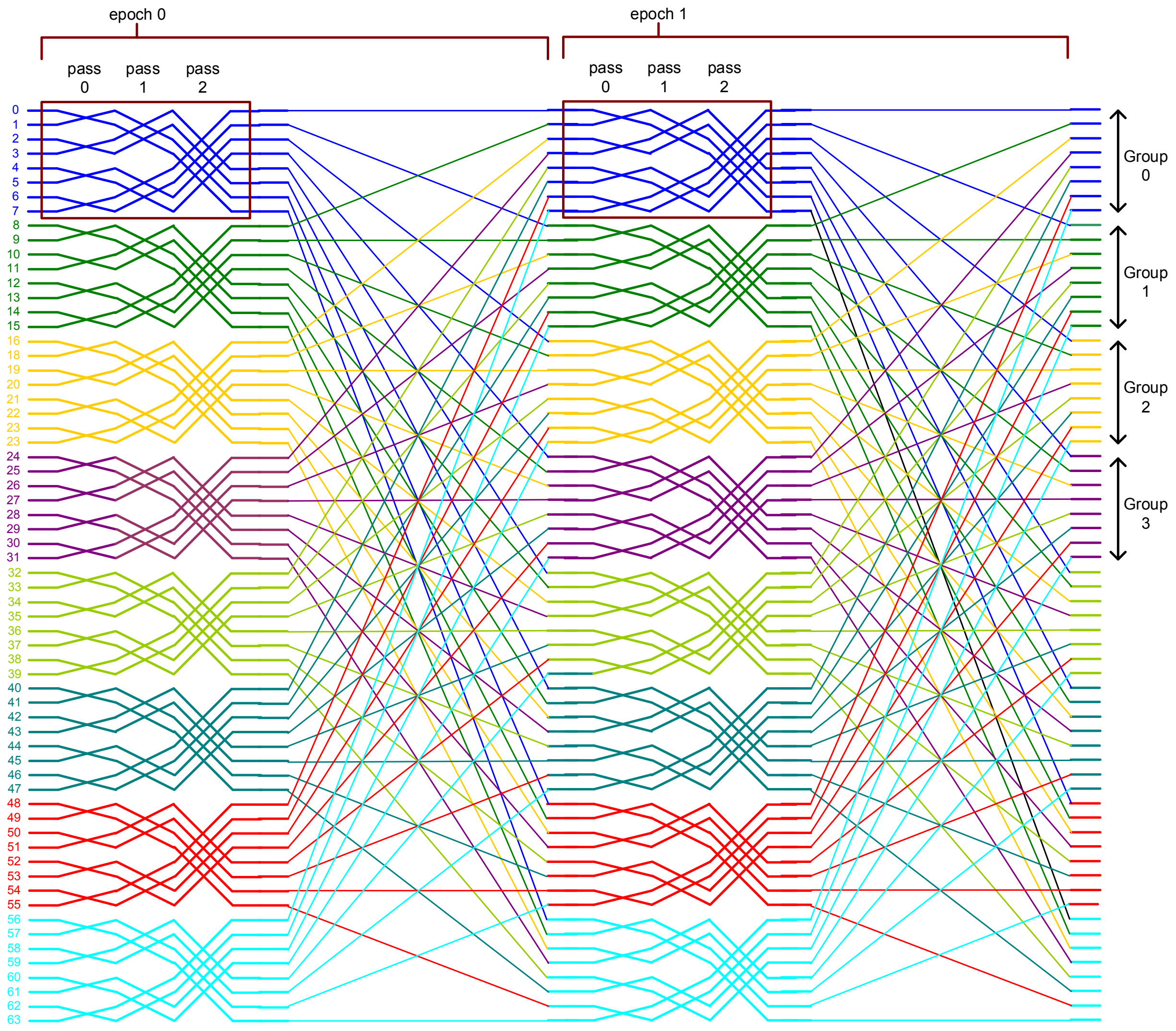


Figure 5. Dataflow for a Cached FFT 64-point DIT [10]

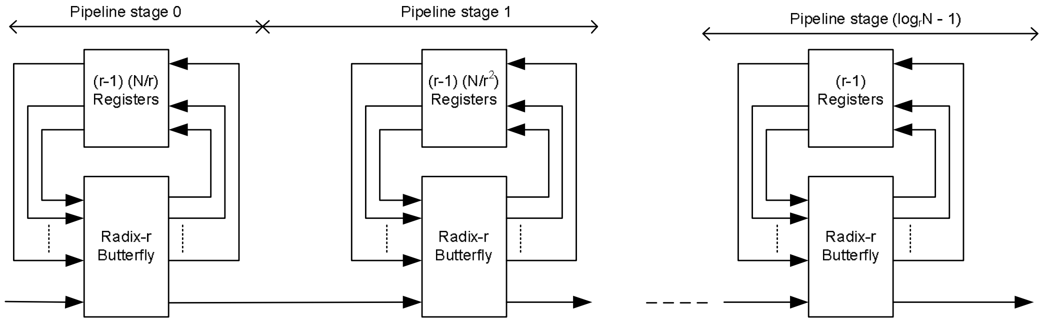
Table 2. Butterfly Addresses, Cache Address and W_N Coefficients for a 64-Point, Radix-2, DIT, 2-Epoch Cached FFT [10]

Epoch Number	Pass Number	Butterfly Address Digits (*= one digit)	Cache Address Digits (*= one digit)	W_N Butterfly Coefficients
0	0	g_2 g_1 g_0 b_1 b_0 *	b_1 b_0 *	W_{64}^{00000}
	1	g_2 g_1 g_0 b_1 * b_0	b_1 * b_0	$W_{64}^{b_0 0000}$
	2	g_2 g_1 g_0 * b_1 b_0	* b_1 b_0	$W_{64}^{b_1 b_0 000}$
1	0	b_1 b_0 * g_2 g_1 g_0	b_1 b_0 *	$W_{64}^{g_2 g_1 g_0 00}$
	1	b_1 * b_0 g_2 g_1 g_0	b_1 * b_0	$W_{64}^{b_0 g_2 g_1 g_0 0}$
	2	* b_1 b_0 g_2 g_1 g_0	* b_1 b_0	$W_{64}^{b_1 b_0 g_2 g_1 g_0}$

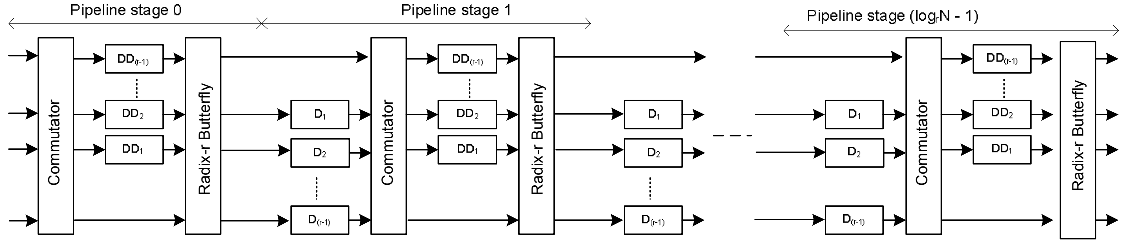
2.3 Pipeline FFT

Groginsky and Works developed an early pipeline FFT design [2]. Later, several pipeline architectures were proposed and designed [3]-[5]. The radix-r pipeline FFT consists of $\log_r N$ pipeline stages. The pipeline processes $\log_r N$ butterflies in parallel. Consequently, the radix-r pipeline FFT has as a speedup of (at least) $\log_r N$ compared to an FFT performed on a single radix-r FFT processor. Based on the number of paths between the pipeline stages, FFT pipelines are classified into the Single-path Delay Feedback (SDF) and the Multi-path Delay Commutator (MDC).

The SDF pipeline has one path (i.e., intermediate result) between the pipeline stages. Figure 6 illustrates the SDF pipeline for a radix-r N-point pipeline FFT. Each pipeline stage consists of a radix-r FFT butterfly and delay elements to hold intermediate values. In each pipeline stage, the delay element output and previous pipeline stage output are fed into the butterfly. For pipeline stage_i, the size of the delay elements is $(r-1)(N/r^{(\text{pipeline\_stage}+1)})$ delay elements. The total number of delay elements in the MDC pipeline is N-1. Since the pipeline processes one point in each clock cycle, the pipeline throughput is one point per cycle.



Radix-r SDF Pipeline FFT



Radix-r MDC Pipeline FFT

Figure 6. SDF and MDC Radix-r FFT Pipelines

The radix-r MDC pipeline FFT utilizes *r* paths between pipeline stages, as shown in Figure 6 [3] and [4]. Each pipeline stage receives *r* intermediate results from the previous pipeline stage, and passes *r* outputs to the next pipeline stage. A pipeline stage consists of an *r*-input commutator, a radix-r butterfly and two delay element sets. The first set is located before the Commutator (shown as D). This set does not exist in pipeline stage 0. The second set is situated after the Commutator (shown as DD). The size of the j-th element of each set in pipeline stage$_i$ is $Di_j = DDi_j = j \times (N/ r^{i+1})$ delay elements. The total number of delay elements in the MDC pipeline is $(r+1)N/2 - r$. Since, the pipeline processes *r* points in each clock cycle, the MDC pipeline throughput is *r* points per cycle.

Table 3 compares SDF and MDC pipelines features [5]. In summary, the radix-r pipeline FFT has a speedup of (at least) $\log_r N$ compared to an FFT performed on a single radix-r FFT processor. However, because it employs delay elements (i.e., shift registers) the pipeline FFT power dissipation is high.

Table 3. SDF and MDC Comparison [5]

	SDF	MDC
Delay elements	$N-1$	$(r+1)\,N/2 - r$
Butterfly units	$\log_r N$	$\log_r N$
Throughput	1	r

2.4 El-Khashab, *et al.*: the Modular Pipeline FFT

El-Khashab, *et al.* developed the modular pipeline FFT described in [6]-[9]. The N-point modular pipeline FFT consists of two M-point FFT pipeline modules joined by a specialized center element, where $M = \sqrt{N}$. The center element contains coefficient and data memory as well as addressing, routing and control logic. The modular pipeline FFT significantly reduces the number of delay elements. Moreover, the coefficient storage is concentrated within the center element, which can be implemented using energy-efficient RAM memories. Additionally, the throughput of the modular pipeline FFT is identical to that of the standard pipeline FFT, although end-to-end latency of the modular pipeline is slightly higher.

The basic idea of the modular pipeline is very similar to the cached-FFT algorithm. In fact, the modular pipeline can be viewed as a special case of the cached FFT algorithm, where epochs = 2 and $C = \sqrt{N}$. Figure 7 shows an example of 64-point FFT using the modular pipeline algorithm. Clearly, the first $\sqrt{N}$ point pipeline maps to epoch 0 and the second $\sqrt{N}$ point pipeline maps to epoch 1.

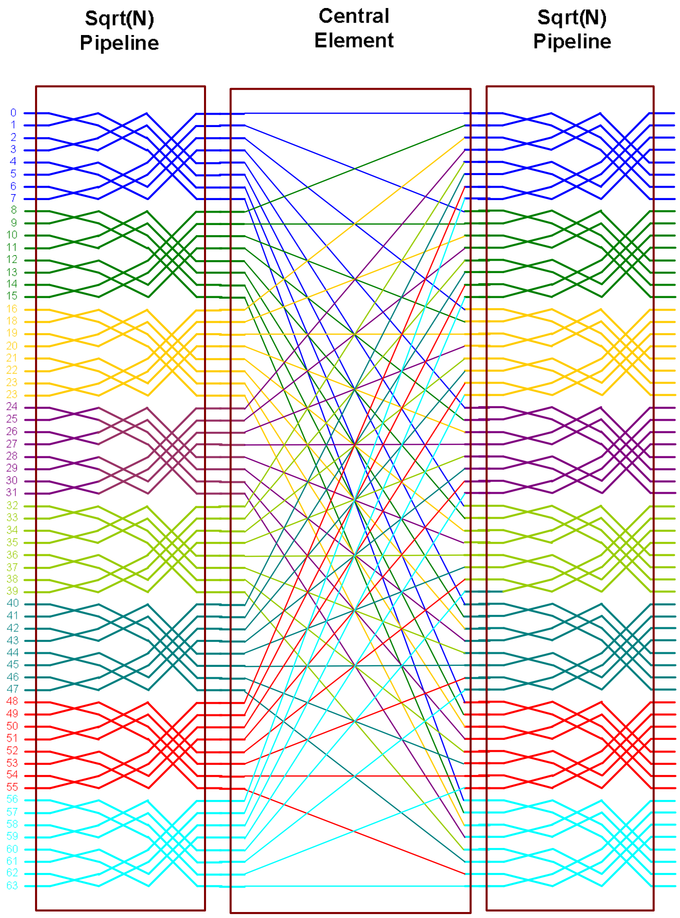


Figure 7. Basic Flow of the Modular Pipeline FFT Using a 64-Point Example

The modular pipeline FFT algorithm can be expressed mathematically in the following equation, which demonstrates the two-module $\sqrt{N}$ FFT:

$$X(\sqrt{N}k_1 + k_0) = \sum_{m_0=0}^{\sqrt{N}-1} W_N^{m_0 k_0} \left(\left(\sum_{m_1=0}^{\sqrt{N}-1} x(\sqrt{N}m_1 + m_0) W_{\sqrt{N}}^{m_1 k_0} \right) \times W_{\sqrt{N}}^{m_0 k_1} \right)$$

$$X(\sqrt{N}k_1 + k_0) = \sum_{m_0=0}^{\sqrt{N}-1} \sum_{m_1=0}^{\sqrt{N}-1} x(\sqrt{N}m_1 + m_0) W_N^{m_1 k_0 \sqrt{N} + m_0 k_1 \sqrt{N} + m_0 k_0}$$

where,

$0 \le k_0, k_1 \le \sqrt{N} - 1$

To obtain the correct results, the transforms of the first module are combined (in a methodic way) and are fed to the second module. In addition, adjustment (i.e., pre-rotation multiplications) must be made for intermediate results prior to the second module, as shown in the following equation:

$$y'(\sqrt{N}k_0 + k_1) = y\ (\sqrt{N}k_0 + k_1) \times W_N^{k_1 k_0}$$

The above mathematical manipulation can be expressed in a butterfly diagram. Figure 8 shows a butterfly diagram with intermediate values for a 16-point FFT. The first module performs the first two stages of the FFT algorithm and has four identical and independent FFTs. Similarly, the second module, which performs the last two stages of the FFT algorithm, has four independent FFTs. It is possible to construct the second module with the same first module FFTs, as shown in Figure 9. This demonstrates that the N-point FFT is now divided into two $\sqrt{N}$ point FFTs. However, each intermediate value must be pre-rotated by a specific W^k_N coefficient. The second module inputs can be described by an FFT number, g and a sample number, s. The FFT number describes which second module FFT processes an input. The sample number, s indicates a unique input number within a particular FFT. The amount of rotation for pre-rotators increases predictably for intermediate values y_0 to y_N. The pre-rotation required is given by:

$$y'_k = -y_k W_N^{gs}$$
$$0 \le g, s \le M - 1,$$
$$M^2 = N$$

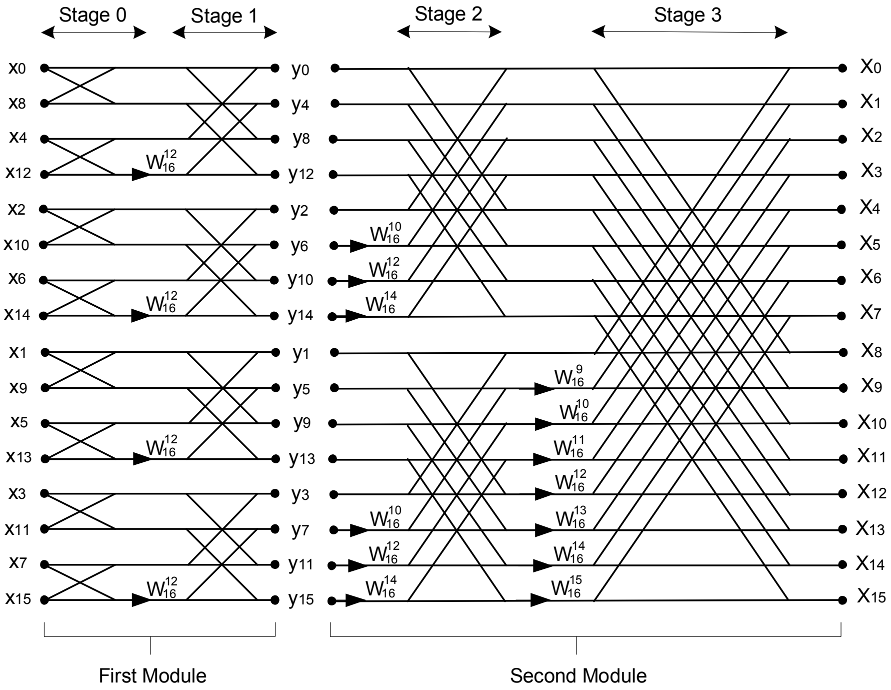


Figure 8. 16-Point FFT Butterfly Diagram with Intermediate Values

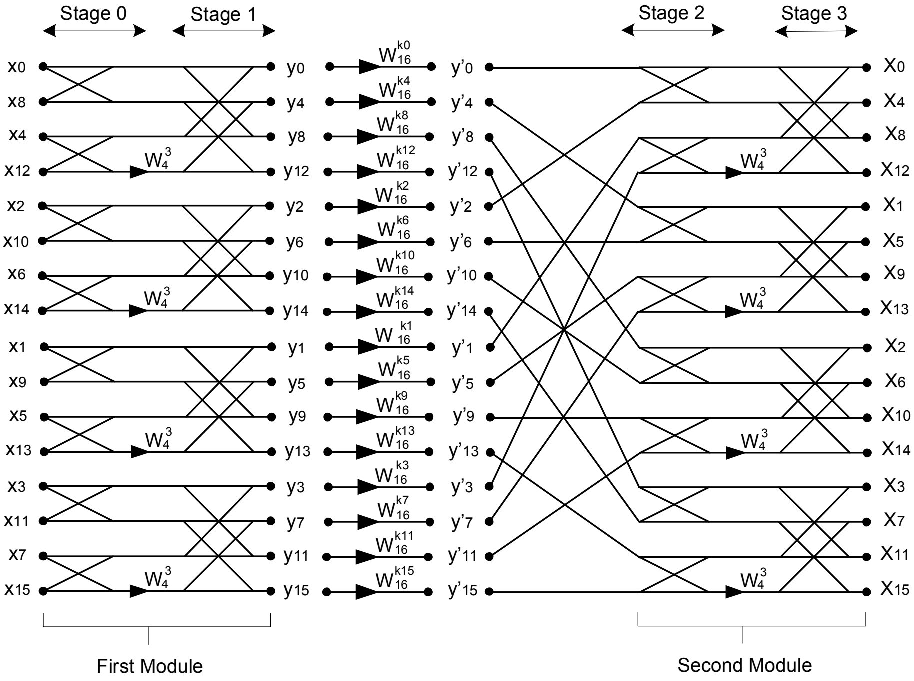


Figure 9. 16-Point FFT Butterfly with Identical First and Second Modules

Radix-2 and radix-4 architectures have been proposed for the modular pipeline FFT. The remainder of this section will examine the radix-2 architecture. Figure 10 shows the overall architecture of the radix-2 modular pipeline FFT. It consists of the two M-point FFT modules and a center element. The center element includes an address generator, RAMs for storing intermediate values and ROMs for the coefficients. The design allows data to be both read and written simultaneously to maximize performance. The operation can be explained as follows. Two discrete inputs are received from the left side of the pipeline. The address generation guarantees the two points have different parities, and hence they reside in different memories. Once N points have been output from the first module, the control dispatches intermediate data to the second module. At the same time, the next *N* points begin entering the first module. Hence the pipeline is able to input and output data on every clock cycle.

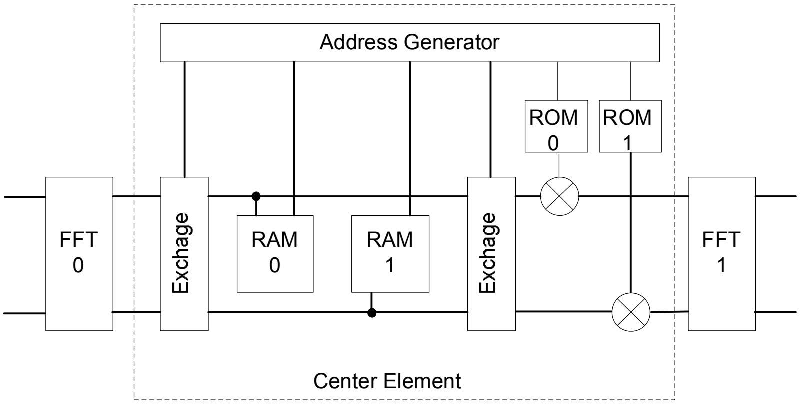


Figure 10. Radix-2 Modular Pipeline Architecture

The address generation logic ensures that inputs are ordered in a specific way in the RAMs, enabling the center element to read (data to FFT1) and write (data from FFT0) data to the same address on each cycle. Table 4 shows the address sequence for a 16-point example. Input address generation logic, shown in Figure 11, utilizes a $\log_2 N$-bit counter. For the radix-2 design, the counter is cleared after two transfers. Further, the "exchange" signal is fed to the exchange block in Figure 10. For DIT, the input address is obtained by a ($\log_2(N)/2$)-1 left shift of the counter output. The top bit of the address can be 0 or 1 depending on the input port. Data in ROMs are ordered in a specific way, so no address generation is required. The output address generation logic, illustrated in Figure 12, partitions the low $\log_2(N)$-1 bits of the counter into a_0, a_1 and a_2 sections, where the size of a_0 and a_1 is ($\log_2(N)/2$)-1.

Table 4. Radix-2 Address Sequence for Two 16-Point FFTs

FFT	Signal								
0	Clock	0	1	2	3	4	5	6	7
	RAM_0	0	1	2	3	4	5	6	7
	RAM_1	0	1	2	3	4	5	6	7
	EXCH	0	0	0	0	1	1	1	1
1	Clock	8	9	10	11	12	13	14	15
	RAM_0	0	2	1	3	4	6	5	7
	RAM_1	4	6	5	7	0	2	1	3
	EXCH	0	0	0	0	1	1	1	1

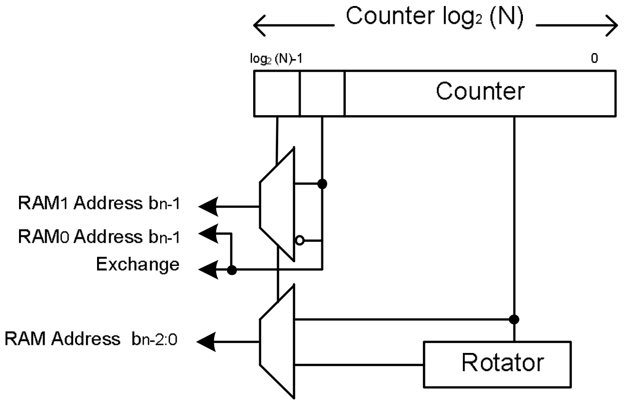


Figure 11. Input Address Generation Logic

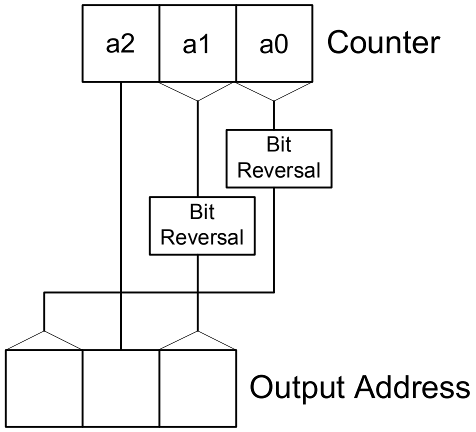


Figure 12. Output Address Generation Logic

Hardware Complexity:

The conventional (standard) radix-r pipeline FFT requires (N-r) delay elements and the same number of coefficients. Since the modular pipeline FFT consists of two $\sqrt{N}$ point pipeline FFT modules, it requires 2($\sqrt{N}$ -r) delay elements and the same number of coefficients (ROM). The central element requires an N-word RAM for pre-rotation results. Hence, the modular pipeline uses more memory. However, most of its memory is concentrated in the central element, which is implemented with energy-efficient RAMs. In contrast, delay elements (i.e., shift registers) are often realized with less energy efficient flip-flops. Also, there is an extra complex multiplication in the central element.

Time Complexity:

For the conventional (standard) pipeline FFT, it takes $T = 2(N / r)$ cycles to perform an N-point FFT. For the modular pipeline FFT, it takes $T = 2(\sqrt{N} / r)$ cycles to complete the two pipeline FFTs. The central element accumulates data from the first r-points of the first module before sending data to the second module. Therefore, the total delay for the modular pipeline FFT is:

$$T = 2\left(\frac{\sqrt{N}}{r}\right) + 2\left(\frac{N}{r}\right) - 2 \approx \frac{2}{r}\left(N + \sqrt{N}\right)$$

Clearly, the modular pipeline FFT has slightly higher latency than the standard pipeline FFT. Table 5 summarizes the hardware and timing comparisons.

Table 5. Complexity of Radix-r Standard and Modular Pipeline FFTs

	Standard	**Modular**
ROM (Coefficient)	N-r	2($\sqrt{N}$ -r)
Delay Elements	N-r	2($\sqrt{N}$ -r)
Butterflies	log_rN	log_rN
CE RAM	0	N
Throughput	r points / cycle	r points / cycle
Delay	$2\left(\frac{N}{r}\right)$	$\frac{2}{r}\left(N+\sqrt{N}\right)$

In summary, the modular pipeline FFT provides a method to calculate *N*-point FFTs using two $\sqrt{N}$ point FFTs. Despite the fact that it requires more memory; it has fewer delay elements. The modular pipeline FFT requires an additional pre-rotation multiplication and has a slightly higher latency than the standard pipeline FFT.

2.5 Zhong, *et al.*: the Processor Ring Architecture

In [15] and [16], Zhong, *et al.* proposed a power-scalable reconfigurable ring-architecture multiprocessor for a single chip FFT/IFFT processor. The processor is capable of processing different FFT sizes (8-point to 4096-point) with scalable power across FFT sizes. This is accomplished by subdividing the large *N*-point FFT into three passes of smaller FFTs: *r1*-point, *r2*-point and *r3*-point, which creates a logical pipeline of three stages. Further, the FFT processor consists of: a four-processor ring (performs a small size FFT with an equivalent efficiency of one butterfly per clock), a four-point FFT co-processor (performs a 4-point FFT with an equivalent efficiency of one butterfly per clock), an angle rotator (performs phase rotation multiplications), RAMs, an address generator and a block floating point unit. The remainder of this section focuses on the four-processor ring.

The processor ring, shown in Figure 13, uses a ring topology. In addition, processors communicate with one another through on-chip dual-ported register files placed between adjacent pairs of processors. The advantages of the ring architecture are [15], [16]:

- It avoids accessing off-chip RAMs,
- It eliminates memory-access contention,
- It requires no bus arbitration.

Each processor's ALU consists of a multiplier, coefficient RAM, and adders. As a result, each processor is capable of executing the following operations in one cycle: multiply, data-move or two adds. Hence, the processor ring can execute up to 12 computations per cycle: [4 processors × (1 multiply + 2 adds)].

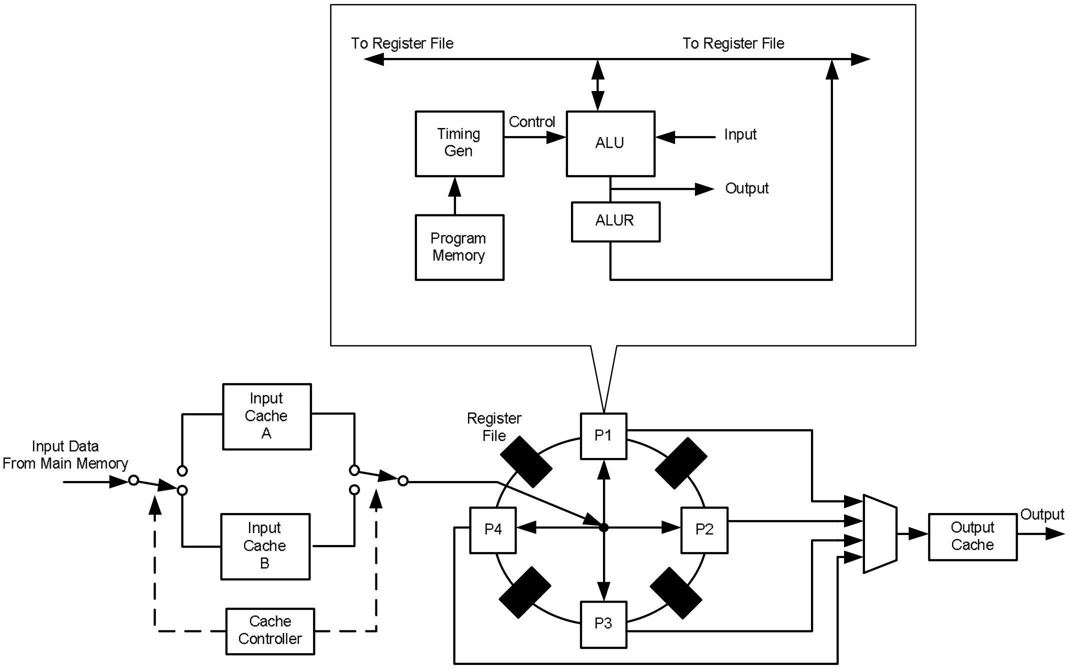


Figure 13. Processor Ring

Example of 4-point FFT

Recall the operations for a radix-2 DIT FFT butterfly are expressed as:

$$X_m = X_{m-1} + W_N^k Y_{m-1}$$
$$Y_m = X_{m-1} - W_N^k Y_{m-1}$$

where *X, Y* and *W* are complex numbers that can be expressed as:

$$X_m = Xr_m + jXi_m$$
$$Y_m = Yr_m + jYi_m$$
$$W_N^k = Wr^k + jWi^k$$

To utilize the processors' resources efficiently, the first step is to reorganize the FFT operations as illustrated in Table 6. Next, the operations are assigned to processors, in this example, P3 and P4, as shown in Table 7. Further, P4 computes the real parts of X_m and Y_m while P3 computes the imaginary parts. The multiplication of Tmp3 occurs in P4 during program step 1, while a previous butterfly's final addition and subtraction are still being completed. The result is stored in the pipeline storage Tmp3. In cycle 2, Xi_{tmp}, Yi_{tmp} and Tmp4 are computed. The final results are computed in the third cycle. Since two butterflies can be processed at once by using four processors, this is equivalent to one butterfly per clock cycle. Higher point FFTs lead to more activities in other processor PEs, which is the basic idea for power scalability. The authors provide power analysis for different values of N, which shows scalable power dissipation for $8 \leq N \leq 4096$. For $N < 256$, the power does not reduce in proportion to N.

In summary, while the use of the processor ring architecture seems to be an interesting idea, the case for using the ring architecture to compute FFTs is weak. The architecture is a better fit for more serialized computation such as FIR filters. Larger values of N require more complex processor programs. Finally, power does scale well for $N < 256$.

Table 6. Calculating X and Y

Real Part of X and Y	Imaginary Parts of X and Y
$Tmp_1 = Yr_{m-1} \times Wr^k$	$Tmp_3 = Yr_{m-1} \times Wr^k$
$Tmp_2 = Yi_{m-1} \times Wi^k$	$Tmp_4 = Yi_{m-1} \times Wi^k$
$Xr_{tmp} = Xr_{m-1} + Tmp_1$	$Xi_{tmp} = Xr_{m-1} + Tmp_3$
$Yr_{tmp} = Xr_{m-1} - Tmp_1$	$Yi_{tmp} = Xr_{m-1} - Tmp_3$
$Xr_m = Xr_{tmp} - Tmp_2$	$Xr_m = Xi_{tmp} - Tmp_4$
$Yr_m = Yr_{tmp} + Tmp_2$	$Yr_m = Yi_{tmp} + Tmp_4$

Table 7. Processor Ring Program Using Two Processors

Cycle	P1	P2	P3	P4
	…	…	…	….
1	…	…	…	….
	…	…	$Tmp_1 = Yr_{m-1} \times Wr^k$	$Tmp_3 = Yr_{m-1} \times Wr^k$
	…	…	$Xr_{tmp} = Xr_{m-1} + Tmp_1$	$Xi_{tmp} = Xr_{m-1} + Tmp_3$
2	…	…	$Yr_{tmp} = Xr_{m-1} - Tmp_1$	$Yi_{tmp} = Xr_{m-1} - Tmp_3$
	…	…	$Tmp_2 = Yi_{m-1} \times Wi^k$	$Tmp_4 = Yi_{m-1} \times Wi^k$
	…	…	$Xr_m = Xr_{tmp} - Tmp_2$	$Xr_m = Xi_{tmp} - Tmp_4$
3	…	…	$Yr_m = Yr_{tmp} + Tmp_2$	$Yr_m = Yi_{tmp} + Tmp_4$
	…	…	…	….

2.6 Cohen: Address Generation Scheme

Cohen designed a method to generate the memory addresses for radix-2 FFTs to prevent memory contention [17]. This is accomplished by having inputs and outputs of each butterfly belong to different memory units. Specifically, the address generator alters the standard sequence of accessing memory locations as illustrated in Table 8 and Table 9.

Table 8. Standard Butterfly Sequence for a 16-Point FFT

Stage	BF0	BF1	BF2	BF3	BF4	BF5	BF6	BF7
0	<0, 1>	<2, 3>	<4, 5 >	<6, 7>	<8, 9>	<10, 11>	<12, 13>	<14, 15>
1	<0, 2>	<1, 3>	<4, 6>	<5, 7>	<8, 10>	<9, 11>	<12, 14>	<13, 15>
2	<0, 4>	<1, 5>	<2, 6>	<3, 7>	<8, 12>	<9, 13>	<10, 14>	<11, 15>
3	<0, 8>	<1, 9>	<2, 10>	<3, 11>	<4, 12>	<5, 13>	<6, 14>	<7, 15>

Table 9. Cohen Butterfly Sequence for a 16-Point FFT

Stage	BF0	BF1	BF2	BF3	BF4	BF5	BF6	BF7
0	<0, 1>	<2, 3>	<4, 5 >	<6, 7>	<8, 9>	<10, 11>	<12, 13>	<14, 15>
1	<0, 2>	<4, 6>	<8, 10>	<12, 14>	<1, 3>	<5, 7>	<9, 11>	<13, 15>
2	<0, 4>	<8, 12>	<1, 5>	<9, 13>	<2, 6>	<10, 14>	<3, 7>	<11, 15>
3	<0, 8>	<1, 9>	<2, 10>	<3, 11>	<4, 12>	<5, 13>	<6, 14>	<7, 15>

Pease observed that the address for each butterfly differs in its parity [18]. Therefore, it is possible to organize the two butterfly points in the memory so that they reside in two different memories. This is accomplished by the address generator design shown in Figure 14. Moreover, the memory addresses (s, t) for the j-th butterfly in the i-th iteration can be expressed as:

$s = \text{ROTATE}_n (2j, i)$

$t = \text{ROTATE}_n (2j + 1, i)$

$n = \log_2 N$

$i = 0, 1, , \ldots (n - 1)$

$j = 0, 1, \ldots , (N/2 - 1)$

$\text{parity} = s[0]\ \textit{XOR}\ s[1]\ \text{XOR} \ldots$

where $\text{ROTATE}_n (X, m)$ rotates the value of X left by m locations.

For example, in stage 2, the values of counters, s and t are illustrated in Table 10. Observe that the final values of s and t differ in the third bit.

Table 10. Cohen Address Generation for Stage Two of a 16-Point FFT

i= counter [4:3]	**j= counter [2:0]**	**s** (before rotate)	**t** (before rotate)	**s** (after rotate)	**t** (after rotate)	**Parity**	**Address MU0**	**Address MU1**
10	000	000**0**	000**1**	0000	0100	0	000	010
10	001	001**0**	001**1**	1000	1100	1	110	100
10	010	010**0**	010**1**	0001	0101	1	010	000
10	011	011**0**	011**1**	1001	1101	0	100	110
10	100	100**0**	100**1**	0010	0110	1	011	001
10	101	101**0**	101**1**	1010	1110	0	101	111
10	110	110**0**	110**1**	0011	0111	0	001	011
10	111	111**0**	111**1**	1011	1111	1	111	101

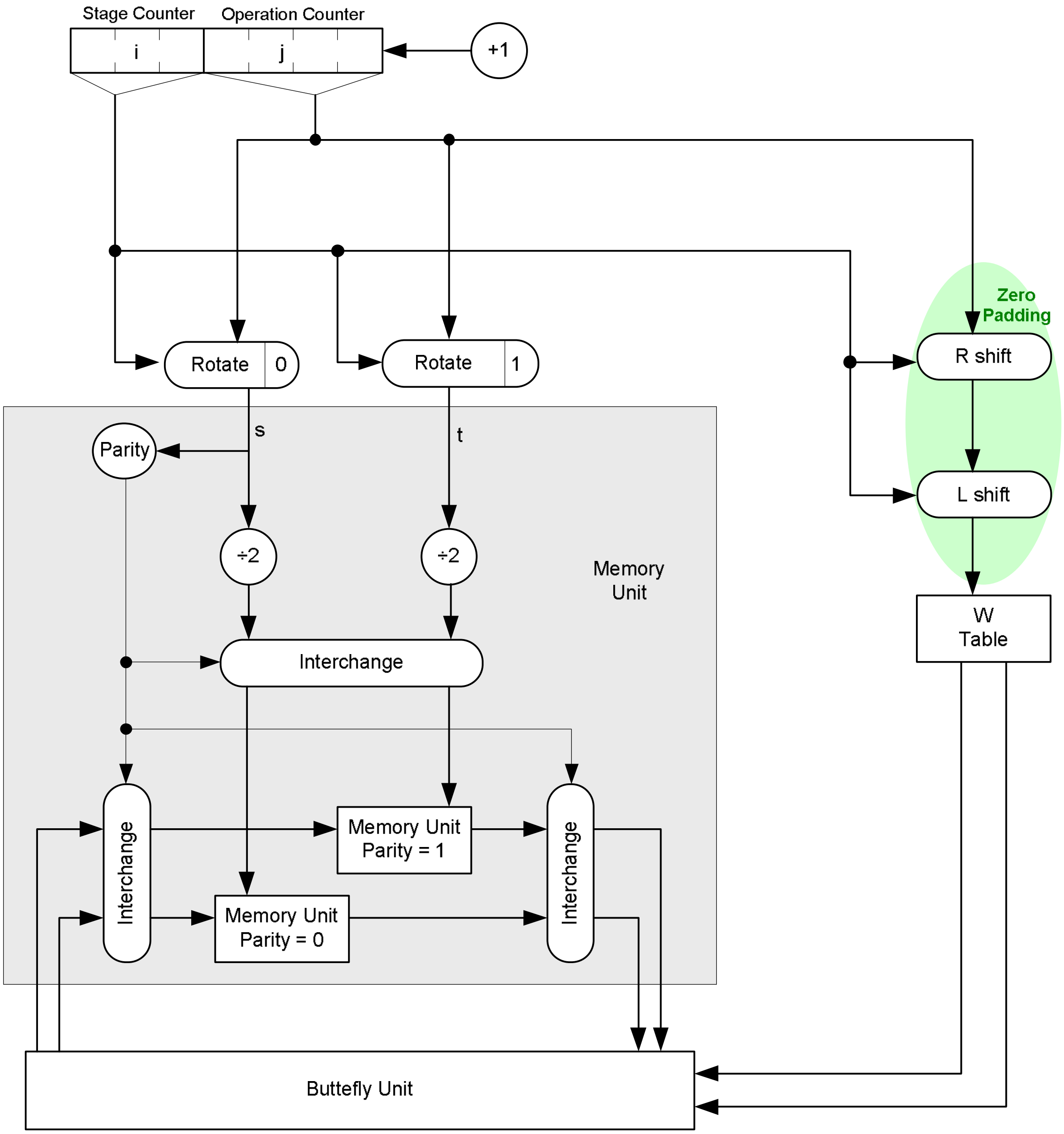


Figure 14. Address Generation Logic [17]

In summary, Cohen's approach resolves memory conflicts in radix-2 FFTs, which enhances the computation speed. Compared to other designs, the design is simple and uses only one counter. Johnson developed a similar algorithm for the general case of a radix-r FFT [19]. On the other hand, Cohen's method performs one butterfly operation per cycle, which limits the performance.

2.7 Ma: Reducing the Complexity and Power of Address Generation

Ma extended Cohen's scheme to optimize hardware complexity, speed and power [20]. Ma proposed an enhanced address generation technique which reduces the delay of address generation to half that of Cohen's method. This is accomplished by the elimination of the interchange operation on Figure 14, resulting in a faster read path. Ma claims that the hardware complexity of this technique is equivalent to that of Cohen's. However, this technique allows both the results of one butterfly to be stored in one memory unit causing memory contention. To resolve the contention, write operations are pipelined, which adds extra hardware cost (i.e., registers, muxes and control logic.)

Ma, *et al.* extended the memory addressing scheme [21]. The proposed scheme utilizes four smaller memory units. A smaller memory is active during an access, which reduces memory power dissipation. In addition, the addressing scheme simplifies the address generation, since the addresses are shared by all memory units. As result, the address generation hardware complexity is reduced by 50%. Further, the address scheme reduces the number of coefficient accesses. However, this method has some disadvantages. It requires complex memory initialization. Also, the method is limited to a specific number of memories (i.e., four) and not a generalized number of memories. Additionally, the scheme does not allow concurrent butterfly processing.

In their first main contribution in the paper, Ma, *et al.* suggested a new *butterfly sequence*. They initially made the following observations (see Figure 15):

1. At stage p, there are 2^p distinct coefficients, where $n = \log_2 N$, and $p = 0, 1, \ldots, n\text{-}1$.

2. Additionally, at stage *p*, the butterflies are divided into $2^{(n-1-p)}$ groups. The coefficients in each group can be classified to two parts: coefficients with address MSB = 0 and coefficients with address MSB = 1. If *C* and *C'* are two coefficients and their addresses *Ac* and *Ac'*, respectively, then:
    a. $C = \mathrm{Re}(C) + j\ \mathrm{Im}(C)$.
    b. If $Ac' = Ac + N/4$, then $C' = -\mathrm{Im}(C) + j\ \mathrm{Re}(C)$, as demonstrated by the following equations:

$$C' = W_N^{AC'} = W_N^{AC+N/4} =$$
$$C' = C \times W_N^{N/4} = C \times (-j)$$

    c. Therefore, *C'* can be generated from real and imaginary parts of *C*.

Using the above two observations and by reordering the sequence of butterflies, it is possible to reduce the number of coefficient accesses by reordering the butterfly sequence as demonstrated in Table 11. Observe that coefficients with address $Ac' = Ac + 4$ follows coefficients with address *Ac*.

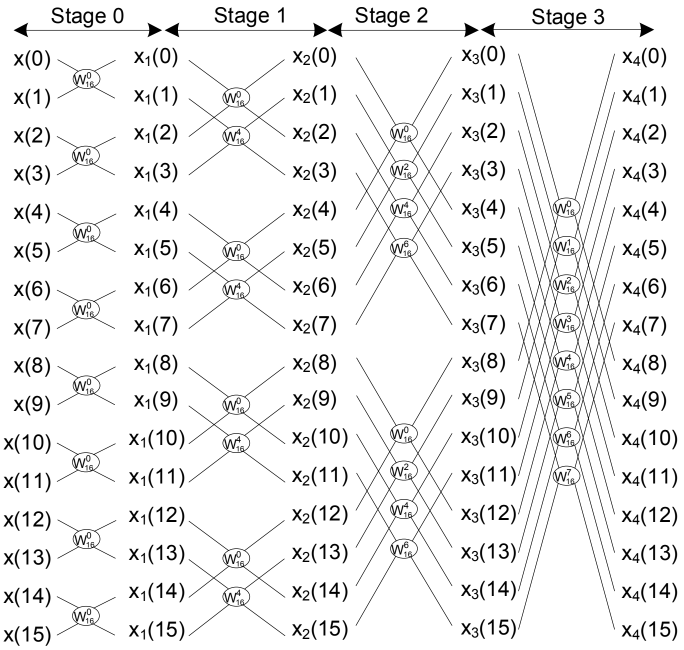


Figure 15. Coefficients in 16-Point DIT FFT Calculations

Table 11. Reordered Butterflies for 16-point FFT

Stage	BF0	BF1	BF2	BF3	BF4	BF5	BF6	BF7
0	<0,1,0>	<8,9,0>	<2,3,0>	<10,11,0>	<4,5,0>	<12,13,0>	<6,7,0>	<14,15,0>
1	<0,2,0>	<1,3,4>	<4,6,0>	<5,7,4>	<8,10,0>	<9,11,4>	<12,14,0>	<13,15,4 >
2	<0,4,0>	<2,6,4>	<8,12,0>	<10,14,4>	<1,5,2>	<3,7,6>	<9,13,2>	<11,15,6>
3	<0,8,0>	<4,12,4>	<1,9,1>	<5,13,5>	<2,10,2>	<6,14,6>	<3,11,3>	<7,15,7>

In the second contribution in the paper, Ma, *et al.* proposed a new *memory partition*. By dividing the memory into four banks and only enabling the memory banks where the butterfly operands and results reside, then nearly half of the memory power consumption can be saved.

The scheme is based on using two basic addressing counters, listed in Table 12. Other counters/variables are derived from the pass counter and the butterfly counter. The addresses for the data and the coefficients are generated as follows:

$A_s(r) = A_t(r) = \mathrm{ROTATE\_LEFT}\ (2B_r, p)$

$A_s(w) = A_t(w) = \mathrm{ROTATE\_LEFT}\ (2B_w, p)$

Memory bank selection is illustrated in Table 14. The scheme requires an initial data assignment to the memories. The memory selection during the initial data assignment is illustrated in Table 14, the bank address is defined as: d_1, d_{n-2}, … d_3, d_2. Similarly, the address for the final output data is defined as: d_{n-3}, … d_1, $d_{n-2.}$

The address generation for read and write operations are illustrated in Figure 16 and Figure 17. Observe the following key features:

- The generated addresses are shared by memory modules which reduces hardware complexity,
- Parity logic is not used (unlike Cohen's scheme) which speeds up the write-path, read paths and address generation path,
- Coefficient address generation is simplified (not shown in the figure.)

Ma, *et al.* claim that the memory access is faster that Cohen's scheme. They further conclude that their scheme hardware cost is 50% less than Cohen's. The power consumption is less because:

- A smaller memory is active during an access,
- The address generation scheme is simplified,
- There is less coefficient access.

Table 12. Counters and Variables Used in Address Generation

Counter	Definition
Pass counter	$P = p_{(\log_2 n)-1} \cdots p_1 p_0$
Butterfly counter	$B = b_{n-2}, \ldots b_1, b_0$
Reconstructed butterfly counter	$B_{fly} = b_0, b_{n-2}, \ldots b_1$
Butterfly counter variable for read operation	$B_r = b_1, b_{n-2}, \ldots b_2$
Butterfly counter variable for write operation	$B_w = b_0, b_{n-2}, \ldots b_2$
Addresses for butterfly data in read operation	$A_s(r)$, $A_t(r)$
Addresses for butterfly data in write operation	$A_s(w)$, $A_t(w)$
Sample Index (for input and output indexing)	$D = d_{n-1}, d_{n-2}, \ldots d_1, d_0$

Table 13. Memory Partition and Data Assignment

Read/Writes	Decoding		Memory
Reads	$0b_0$	00	M_{00}
		01	M_{01}
	$1b_0$	10	M_{10}
		11	M_{11}
Write	$b_1 0$	00	M_{00}
		10	M_{10}
	$b_1 1$	01	M_{01}
		11	M_{11}
Input/Output	$d_0 d_{n-1}$	00	M_{00}
		01	M_{01}
		10	M_{10}
		11	M_{11}

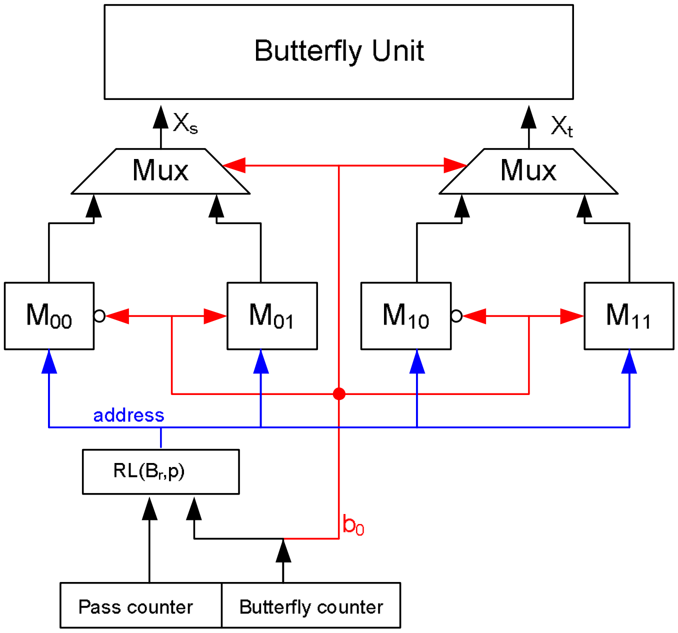


Figure 16. Read Address Generation

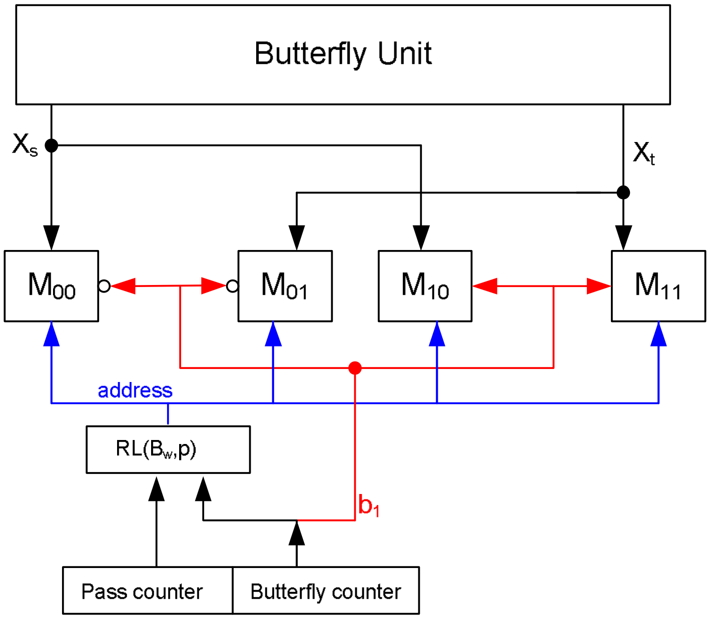


Figure 17. Write Address Generation

2.8 Summary

In this chapter, several FFT processors have been discussed. Table 14 summarizes the pros and cons for each processor. There is a need for a new approach to address the following issues:

- Exploit parallel butterfly processing for faster FFTs,
- Provide an architecture to efficiently connect butterflies and memories,
- Eliminate memory contention for any number of memories,
- Utilize only memory arrays (i.e., RAMs) for energy efficiency,
- Avoid FFT decomposition to avoid the latency and power penalty of extra rotational multiplications,
- Facilitate power scalability and FFT-size configurability.

Table 14. Summary of the Approaches

Scheme	Advantages	Disadvantages
Cached-FFT	• Increased clock frequency • Energy efficiency	• Overhead of cache and cache controller • Lacks configurability for small FFT • Lacks scalability for power
Pipelines (SDF, MDC)	• Increased clock frequency • Speed up by (at least) $log_r N$	• Increased power due to switching power in delay elements • Limited parallelism per stage
Modular Pipeline	• Replaces N-pipeline with two $\sqrt{N}$ pipelines • Reduces storage for delay elements and coefficient • Implements the central element memory using energy-efficient RAMs	• Additional pre-rotation multiplications due to decomposition add to latency and power • Larger overall memory
Ring Architecture	• Configurability and Power scalability • Computations are pipelined/paralleled • Data can be programmed to flow clockwise/counterclockwise/both direction • No bus arbitration is required	• No clear advantage of using ring topology in FFT design • Additional pre-rotation multiplications due to decomposition add to processing delay and power • Complex control • For $N < 256$, the power is not reduced in proportion to N.
Cohen	• Resolves memory conflicts in radix-2 FFTs, which enhances the computation speed • Simple hardware: one counter	• Performs one butterfly operation per cycle, which limits the performance
Ma, *et al.*	Compared to Cohen's: • Memory access is faster. • Hardware cost is 50% less • Power consumption is reduced	• It does not allow concurrent butterfly processing • Method requires fixed number of memories (i.e., four) • Complex memory initialization is required

Chapter 3

Research Methodology: Approach and Flow

This chapter presents the overall research strategy. It outlines the general approach of developing the architecture and algorithm from an idea to the full implementation. Next, it discusses the design flow and the simulations performed at various design stages.

3.1 Research Approach

Figure 18 demonstrates the research approach. Initially, selected FFT publications were examined. Concurrently, the initial idea of utilizing a switch fabric to design a FFT processor was investigated. The fusion of previous design pros-and-cons and the switch idea was the basis for the switch-based architecture and also for the general contention-free algorithm. Both pipelined and non-pipelined approaches for the switch-based design were studied and published in [22] and [23]. Next, the algorithm to resolve memory contention was developed in two phases. The initial algorithm addressed the case when the number of PEs is two, which was very limiting. As a result, the general algorithm for PE ≥ 2 was developed.

A comparative analysis was conducted on the pipelined and non-pipelined form. Next the implementation started by developing the register transfer level (RTL) Verilog code. Once the RTL was functionally verified, the physical design implemented. Power estimation and reduction were emphasized during the design. Various power reduction techniques, in particular clock gating, were examined and implemented. Clock gating is one of the main techniques to reduce dynamic power [24].

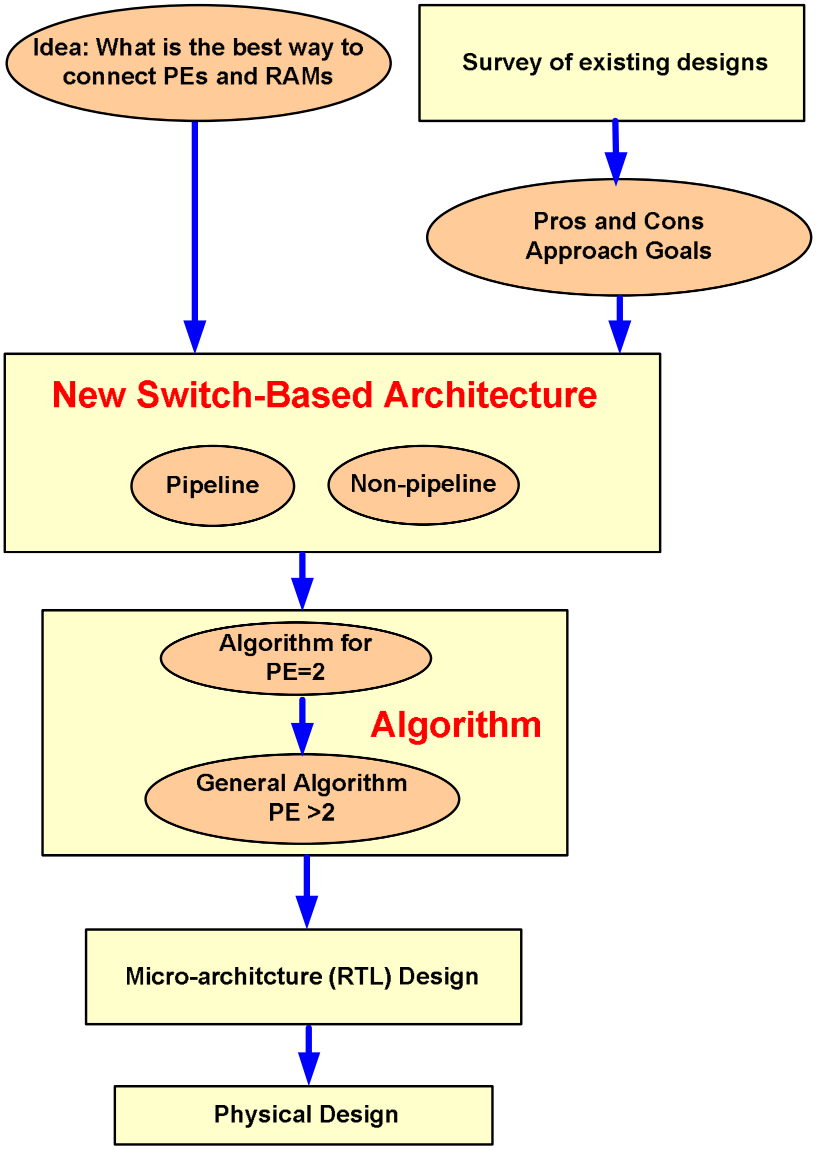


Figure 18. Research Approach

3.2 Design Flow

The overall design flow is shown in Figure 19. The flow starts with the architectural specification and finishes with design layout. The behavioral model (written and verified in Matlab) serves as the golden model for verification.

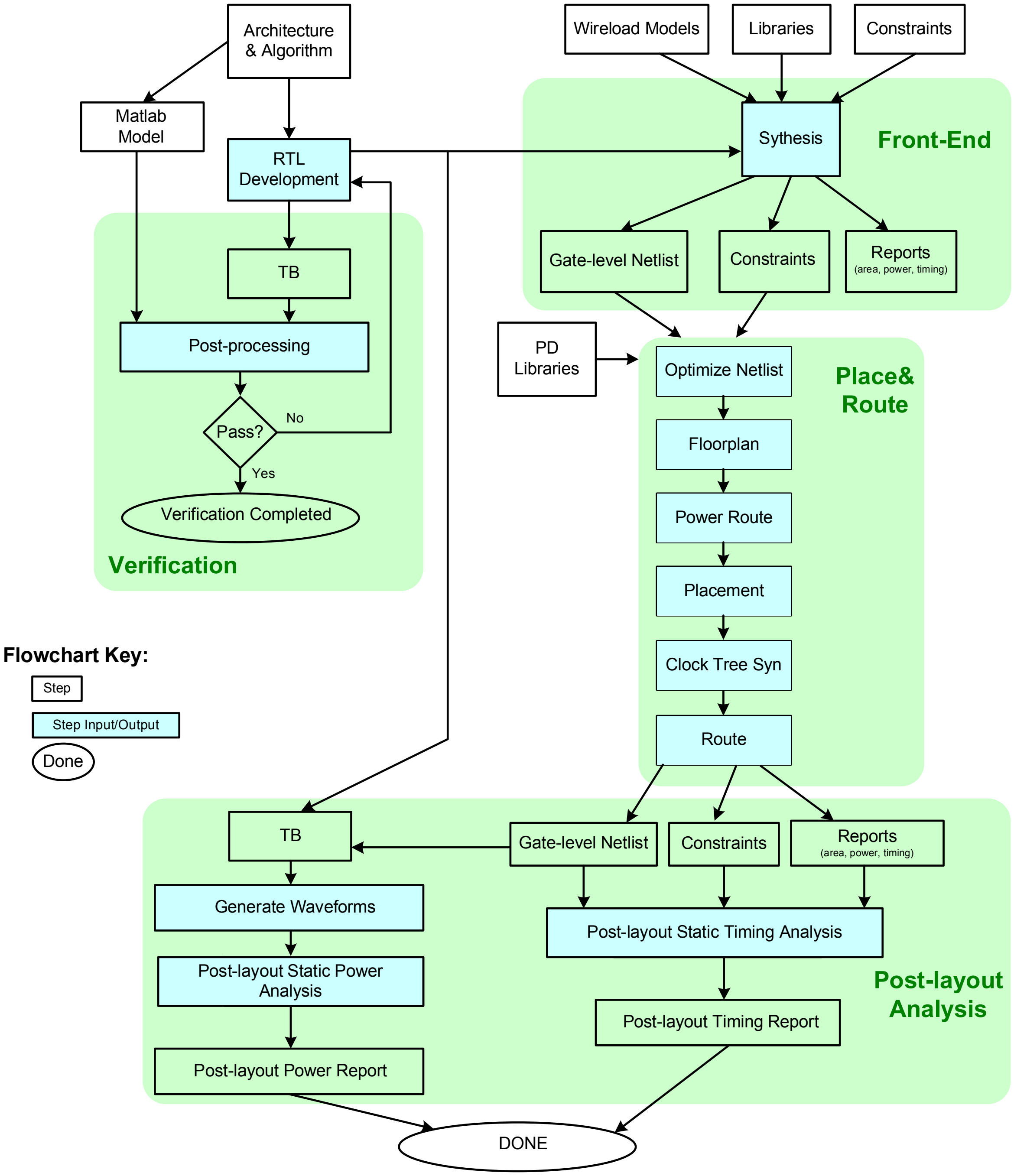


Figure 19. Design Flow

The **RTL verification** validates RTL behavior against the Matlab model using dynamic RTL simulation. To accomplish this objective, a Verilog testbench was developed. The testbench consists of:

1. The design RTL code,
2. The external memory model, written in behavioral Verilog,
3. The Verilog tasks for design initialization and configuration, waveform capturing, memory dumping and memory loading,
4. The gate-level model is instantiated in gate-level simulations.

For each test, the testbench executes the following steps. Initially, the input vectors are loaded in the external memory model. The input vectors represent time-based samples of the processed signal which are generated by the Matlab software. Optionally, a windowing technique (e.g., Hamming) is applied to the input samples. Next, the processor reset is applied, followed by a configuration cycle. The testbench then signals the FFT to begin processing. Once processing is completed, the testbench writes the external memory model contents to an output file. Finally, post-processing scripts are used to compare RTL-generated output files against Matlab-generated results. Because there might be a slight mismatch due to a fixed-point data representation in RTL, the script allows a small error between the results. Optionally, the RTL results are plotted using Matlab plot functions.

The **synthesis step** transforms the RTL code into a mapped gate-level netlist. This step requires the following inputs:

1. The verified RTL,
2. The standard-cell libraries which contain the timing, area and power information for each standard cell,
3. The RAM library which specifies the RAM timing, power and area,
4. The design constraints which specifies the design area, timing and power targets,
5. The wireload model which is used to estimate wire resistance and capacitance based on gate fan-outs. It is created from past designs/test-chips.

One example of wireload model is shown in Table 15.

The output of the synthesis step is the mapped gate-level netlist and timing/area/power reports. Also, a fine-tuned constraint file is generated to be used in the place and route.

Table 15. Wireload Model Format

Fanout	Average Length	Average_cap	Std_dev	Number of Nets
1	45.5 µm	0.016 pF	0.073	266688
…				
10	135.5 µm	0.0501 pF	0.075	1200

The **place and route** step defines the floorplan, cell placement and wire connection. The main tasks are:

1. *Gate optimization* sizes logic gates and add/remove buffers to improve timing,
2. *Floorplanning* determines the overall area of the core/chip. The memory arrays (i.e., RAMs) cells are placed in their locations. Also, placement regions are defined,
3. *Power grid* definition and construction,
4. *Clock tree synthesis* constructs the clock buffer tree which meets the specified clock skew,
5. *Local and global routing*,

Once the routing is completed, the layout tool generates the final gate-level netlist, extracted net RCs and post-layout constraints files. Those files are used in the post timing and power analysis.

Power estimation is done after the synthesis and layout steps. The general flow is shown in Figure 20. The process starts by defining the benchmark tests used in the power flow. Usually, only a portion of the benchmark test is analyzed. This is because the tests are usually lengthy and some of the test segments (e.g., initialization) are not interesting.

Next, the design is simulated and waveforms are captured. The waveforms indicate how often a design net/node is toggling. The power tool uses the activity factors (extracted from waveforms) and libraries to calculate dynamic and leakage power. The wire capacitance is estimated using the wireload model (during synthesis) or the extracted RC (after place and route).

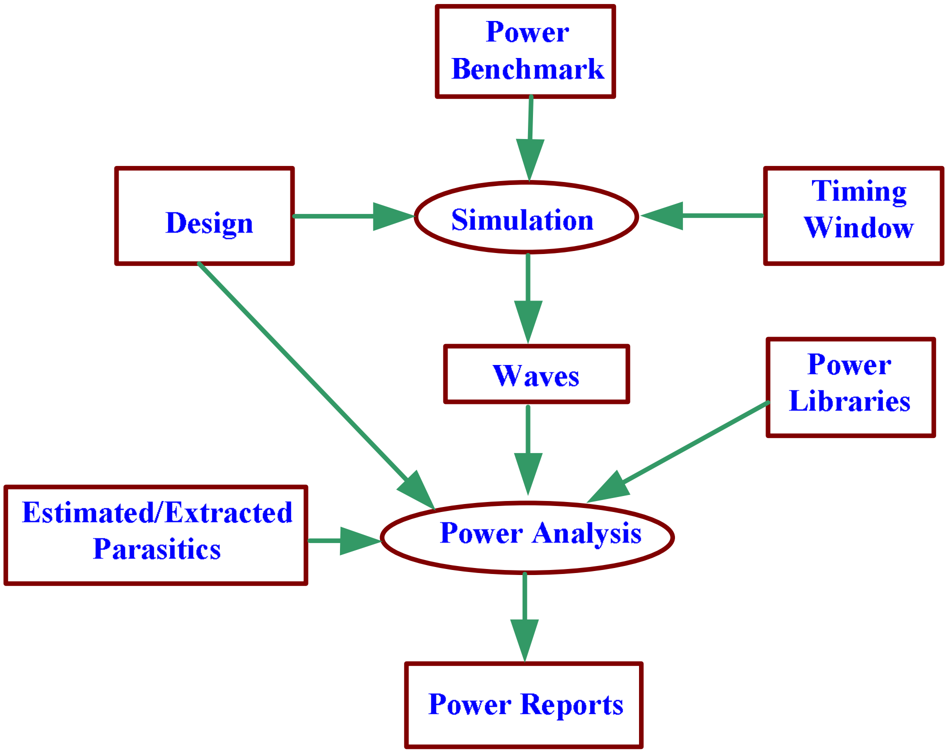


Figure 20. Power Estimation Flow [24]

3.3 Summary

In this chapter, the overall research strategy was described. The research approach and design flow were discussed in detail.

Chapter 4

Architecture and Algorithm

This chapter presents the switch-based architecture and the conflict free algorithm. Two forms of the architecture are examined: pipelined and non-pipelined. A comparison between the two forms is presented, which assists in determining which form is implemented. A 64-point design example is presented to illustrate memory management operations. Finally, the data format and its impact on the dynamic range are examined.

The architectures and algorithm apply to implementations based on decimation-in-frequency (DIF) as well as decimation-in-time (DIT) butterflies. The specifications of stage i in the DIT algorithm is the same as that of stage $\log_2(N)-i$ in DIF algorithm. The following discussion assumes the use of DIF butterflies. Lastly, as mentioned earlier, the dissertation refers to the FFT algorithm stage as "stage" (with no qualification) and refers to the pipeline FFT stage as "pipeline stage".

4.1 Switch-Based Architecture

The radix-2 FFT algorithm consists of $\log_2 N$ stages, where each stage executes $N/2$ complex butterfly operations. The operations in stage i depend on the results from stage $(i-1)$, creating potential data dependencies between the stages. The switch-based architecture exploits parallelism within each stage by utilizing M PEs. Moreover, the architecture can be further extended to a pipelined version to exploit temporal parallelism between stages. The remaining of this section discusses the two architectures.

4.1.1. Non-pipeline Switch-Based [22]

The switch-based architecture connects the heterogeneous elements of the FFT processor using a switch fabric. As a result, data can flow to any component providing maximum data transfer flexibility. The data width is double words; where one word represents the real part and the other word represents the imaginary part of the complex data. The overview of the switch-based architecture is demonstrated in Figure 21.

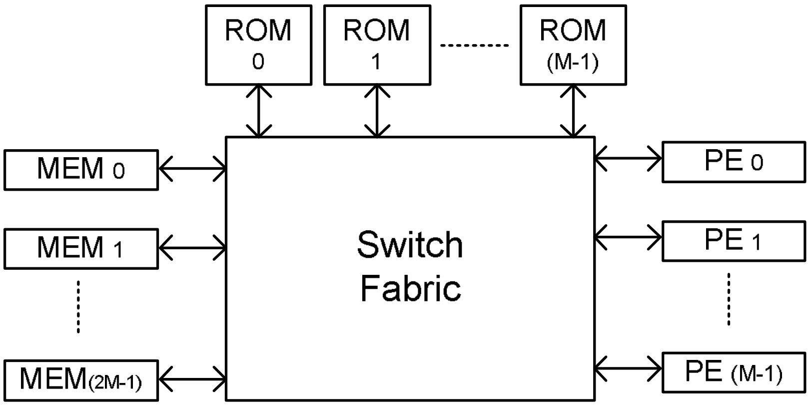


Figure 21. Switch-Based Architecture

The main components of the design are:

1) A switch fabric that connects PEs and memories. An exploded view of the switch is shown in Figure 22. The width of the data buses is double word. The switch consists of the following:
    a) State machine which controls the operation of the switch. It also generates the select MUX select signals and updates the counter values.
    b) MUXes which control the flow of data in the switch based on the MUX select values. MUX **A**, in Figure 22, selects the data feeding port **a** (of PE_k) from the appropriate source RAM. Similarly, MUX **B** generates the data for port **b**. MUX **W** select the appropriate ROM data for port **w**. Lastly, MUX **c** selects the RAM_i write data from the appropriate PE result.

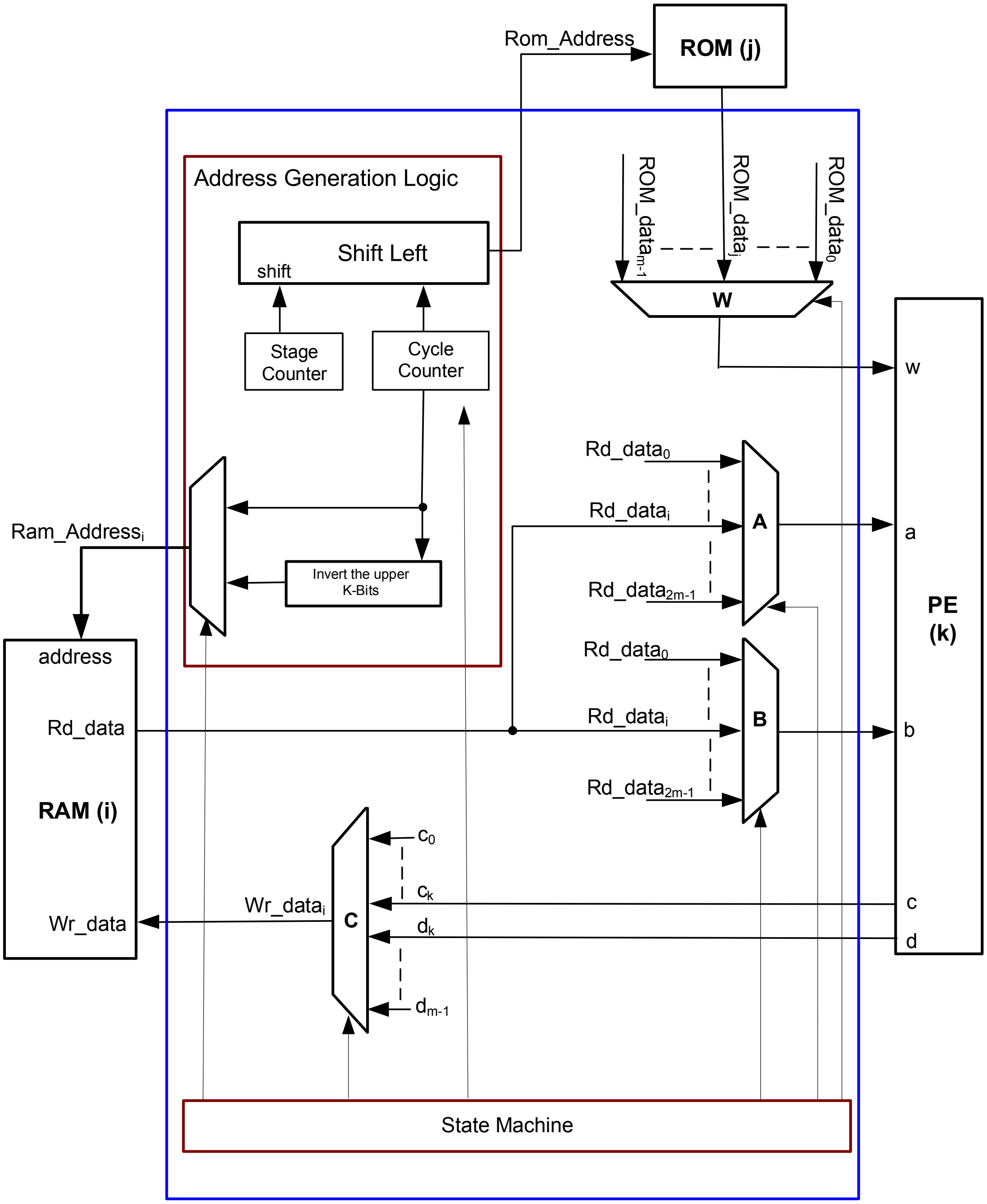


Figure 22. Switch Block Diagram

   c) Address generation logic which creates for the ROMs and RAMs addresses using the stage counter and cycle counter. The stage counter indicates the current stage number in the radix-2 FFT algorithm. In addition, each stage has $N/(2*M)$ cycles. The cycle counter indicates the current cycle number. The ROM address is generated by left-shifting the cycle counter by the value of the stage counter. The RAM address is set either to the cycle counter or the modified cycle counter. The modified cycle counter is generated by inverting a set of the upper bits of the cycle counter. The address generation logic will be discussed in detail in the next chapter.

2) PEs which perform the butterfly operations. Figure 23 illustrates the block diagram of the PE. The PE has three inputs (a, b, w) and two outputs (c, d). Inputs and outputs are complex data (i.e., $a = a_r + ja_i$) and represented by double words. The PE performs the radix-2 butterfly operation:

$$c = a + b$$

$$d = (a - b) * w$$

$$c_r = a_r + b_r$$

$$c_i = a_i + b_i$$

$$temp_r = a_r - b_r$$

$$temp_i = a_i - b_i$$

$$d_r = temp_r * W_r - temp_i * W_i$$

$$d_i = temp_i * W_r - tempr * W_i$$

w is the twiddle factor. The PE consists of six adders and four multipliers, as shown in Figure 23. There are M PEs per stage, where:

- $N/2 \geq M \geq 1$,
- $M = 2^p$, where p is an integer and $p \geq 0$,

3) Memory arrays to store intermediate results. There are $2*M$ memories, and the size of each memory is at least $N/(2*M)$ entries. Each entry is two words wide, to store the real and imaginary parts for a complex data. Memories can be implemented as single-

port RAM, caches or register files, based on the size of the memory and cost constraints,

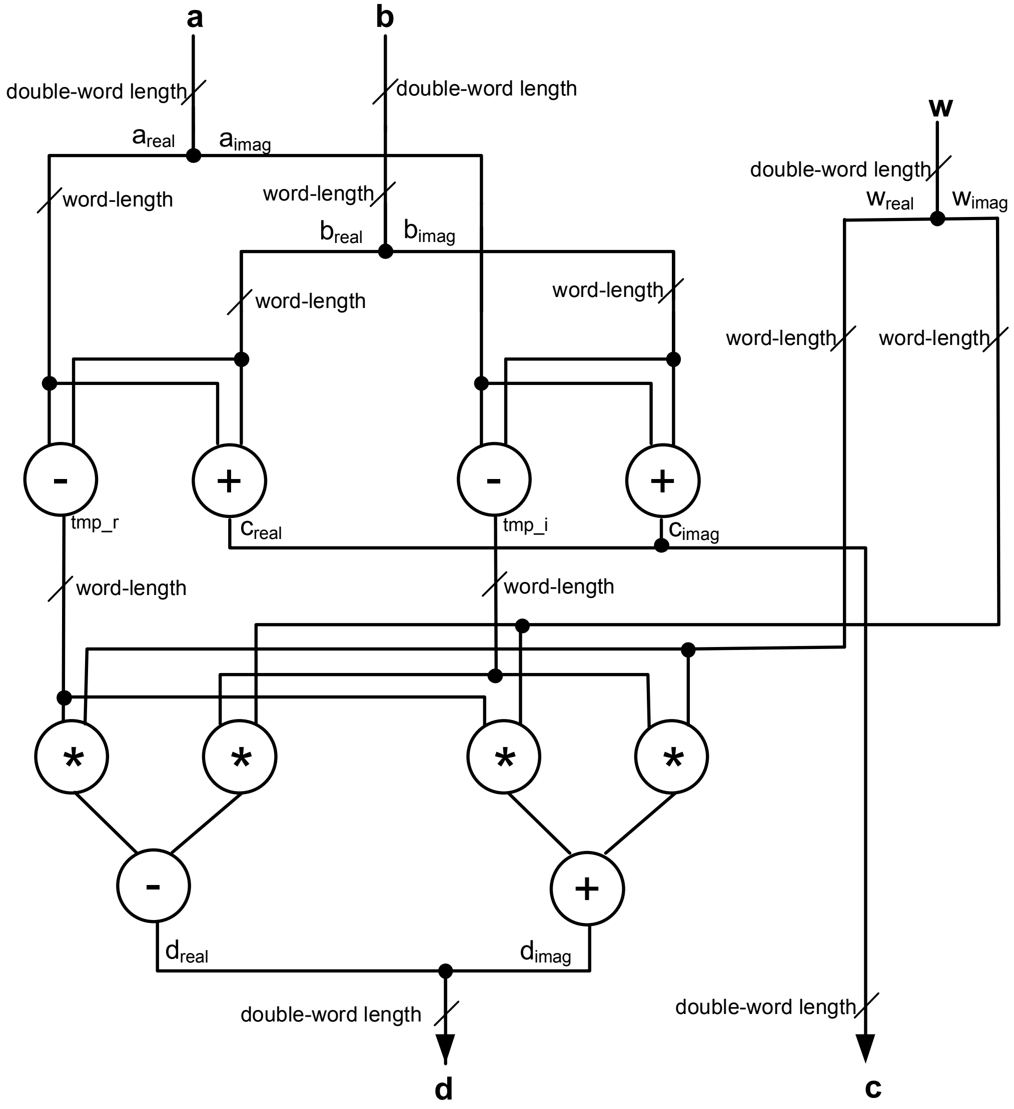


Figure 23. PE Block Diagram

4) Read-Only Memory Arrays (ROMs) to store twiddle factors. Since the twiddle factors do not change, ROMs are very appropriate to store them. There are *M* ROMs per

stage, each with at least *N*/2 entries, where one entry is capable of storing the real and imaginary parts of one twiddle factor.

Figure 24 shows an example for *N*=16 and *M*=2. The architecture is designed to exploit operation-level parallelism in each stage. At each stage, two operations are executed simultaneously by the PEs. Hence, the eight operations of any stage require only four cycles. Mapping a specific operation to any PE is handled by the memory management algorithm which is discussed later.

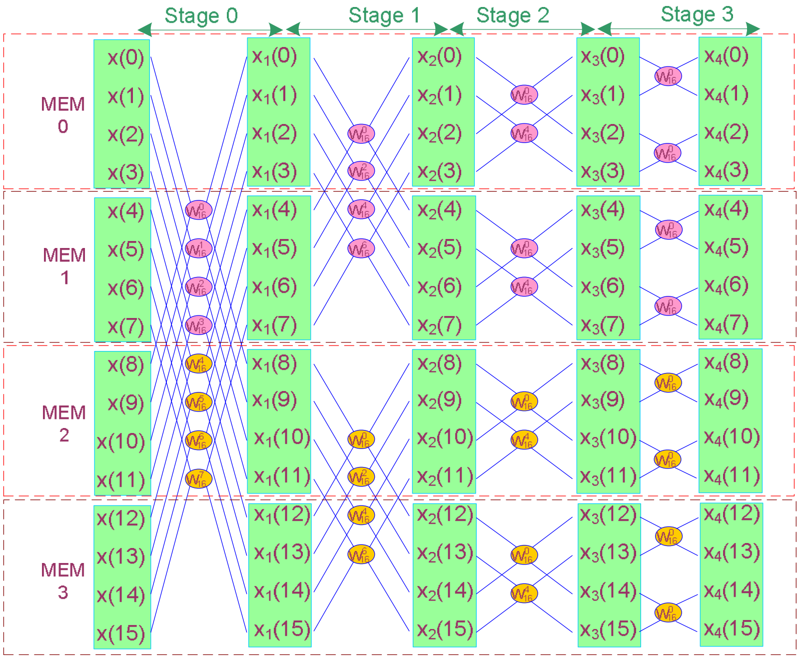


Figure 24. 16-Point DIF FFT

4.1.2. Pipeline approach [23]

The pipeline approach speeds up FFT processing by $M*\log_2(N)$ by cascading $\log_2(N)$ pipeline stages of the basic switch architecture, as shown in Figure 25. The pipeline stages are similar but not identical to the non-pipeline switch. Each pipeline stage consists of:

1. A switch fabric that connects PEs and memories,

2. PEs which have three inputs (a, b, w) and two outputs (c, d) and perform the radix-2 FFT butterfly operation:

   $c = a + b$

   $d = (a - b) * w$

   w is the twiddle factor. There are M PEs per pipeline stage, where:

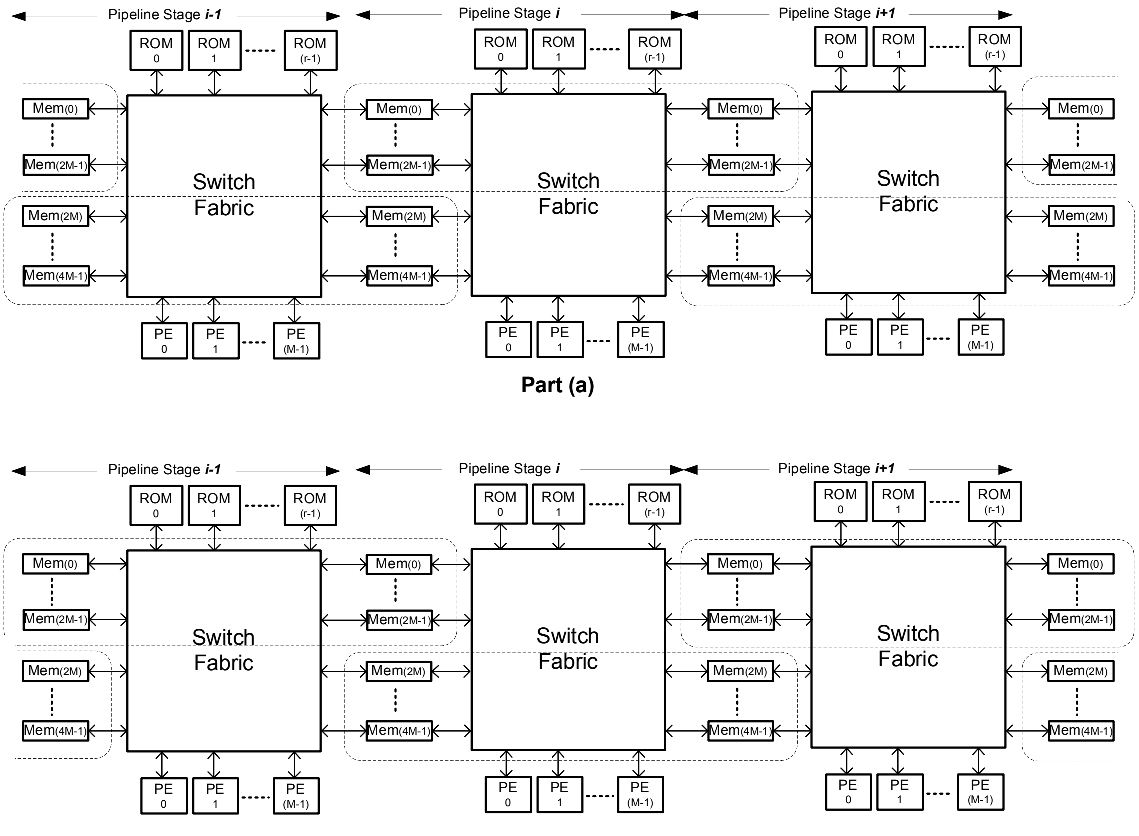
   $N/2 \geq M \geq 1$

   $M = 2^p$, where p is an integer and $p \geq 0$.
3. Memory arrays to store the intermediate results. There are 4*M memories, and the size of each memory is at least N/(2*M) entries. Each entry is two words wide, to store the real and imaginary parts for a complex data. Memories can be implemented as single-port RAM, caches or register files, based on the size of the memory and cost constraints. In each cycle, one set of the memories (Mem_0… Mem_{2M-1}) are written into by the previous pipeline stage PEs. The other set of memories (Mem_{2M}… Mem_{2M-1}) is accessed and by the current pipeline stage PEs. The role of the two sets alternates every stage.
4. M ROMs per pipeline stage to store twiddle factors. ROM size is at least $N/(2*M)$ words. The number and size of ROMs per pipeline stage can be reduced as outlined in Table 16. If the pipeline is designed for a specific value of N, where N is static, the pipeline connectivity and twiddle factors are static. As a result, the design implementation can be optimized since the connectivity of each pipeline stage is predetermined.

Table 16. Number and Size of ROMs per Pipeline Stage

Pipeline Stage "i"	**Number of ROMs**	**Size of ROM**
0	M	N/(2*M)
$\log_2 M \geq i \geq 0$	M	$N/(M * 2^i)$
$i > \log_2 M$	$M/2^{(i-\log2M)}$	1

Each pipeline stage is capable of calculating M radix-2 butterfly results. Using the Instruction Level Parallelism (ILP) classification from [25], the architecture is a superscalar machine with Instruction Parallelism (IP) equal to M. It is also a superpipeline where each pipeline stage has $N/(2*M)$ cycles. The architecture consists of $\log_2(N)$ pipeline stages, where each pipeline stage executes M operations. Therefore, the pipeline speedup can be expressed as: $M*\log_2(N)$. The maximum pipeline speedup is $(N/2)*\log_2(N)$, when $M = N/2$. In this case, memories are reduced to registers, and the switch fabric connects any register to any PE. Clearly, while this case provides the most speed up, its hardware is expensive. The optimal value of M is decided by design parameters: throughput, area and power.



Part (a)

Part (b)

Figure 25. Switch-Based Pipeline

Table 17 summarizes the features of FFT pipeline architectures (surveyed in Chapter 2) and the switch-based architecture. Clearly, the switch-based pipeline offers superior speedup compared to the other pipelines. Moreover, the switch-based pipeline uses energy efficient RAMs instead of delay elements (i.e., registers). On the other hand, the switch based pipeline requires larger memories and more butterfly hardware.

Table 17. Comparing Switch-Based Pipeline FFT with Other Pipeline FFTs

FFT Pipeline	**Butterfly Units**	**Memory Size**	**Speed up**	**Throughput**
Radix-2 Multi-path Delay Commutator	Log_2N	$3N/2 - 2$	$2\log_2 N$	2
Radix-2 Single-path Delay Feedback	Log_2N	$N - 1$	$\log_2 N$	1
Radix-2 Switch-Based Pipeline	$M^* \log_2 N$	$2^*N^* (1+\log_2 N)$	$M^* \log_2 N$	$2M$

4.1.3. Comparing the pipeline and non-pipeline approaches

This section addresses the following questions. "Given fixed target speedup (e.g., S), which factor: the number-of-pipeline-stages or the number-of-PEs should be increased to achieve a more efficient design?" In other words, which is more efficient, the pipeline or the non-pipeline implementation?

To answer these questions, two designs with same speedup are compared. The design parameters are detailed in Table 18. The non-pipeline design has a speed of S since it has S PEs. The pipeline design achieves a speedup of S because it has S pipeline stages, as each pipeline stage has one PE. Table 18 indicates that:

- The non-pipeline total memory is proportional to N, while the pipeline total memory is proportional to $S \times N$. Therefore, the non-pipeline design requires less memory than a pipeline design.
- The complexity of the pipeline switch is constant while the non-pipeline switch is proportional to S^2. Hence, the pipeline switch is simpler than that of non-pipeline.

Therefore, the main disadvantage of increasing the number of pipeline stages in the pipeline design is the increase in total memory. On the other hand, increasing the number of PEs in the non-pipeline design increases the complexity of the switch fabric. Hence, the tradeoffs between the two factors depend on the constraints on the total memory and the maximum complexity of the switch. Because of the larger memory requirement in the pipeline approach, the non-pipeline is chosen to be implemented in this research. Finally, the above analysis does not include the amount of ROM memory required since it is complex and depends on the ROM optimization. The pipeline will use less ROM memory; however, the ROM savings does not offset the extra RAM memory for the pipeline.

Table 18. Comparing Pipeline and Non-Pipeline Designs

Parameter	**Pipeline**	**Non-Pipeline**
Number of Pipeline Stages	S	1
Number of PEs per Stage	1	S
RAM Memory Size	$N/2$	$N/(2*S)$
Number of RAM Memories	$4*(S+1)$	$2*S$
Total Memory	$2*N*(S+1)$	N
Switch Complexity	2*2	$S*S$

4.2 Memory Management Algorithm

Memory conflicts occur when multiple PEs attempt to simultaneously access the same physical memory. In the decimation in frequency FFT, memory conflicts do not occur in the early stages; they occur from stage $\log_2(M)$ to the last stage. In the decimation in time FFT, the conflicts affect stage 0 to stage $\log_2(N) - \log_2(M) - 1$. The 16-point decimation in frequency FFT, shown on Figure 24, demonstrates memory conflicts. Stages 0 and 1 are conflict free, whereas stages 2 and 3 experience memory conflicts. In stage 2 the inputs for the top PE are $x_2(0)$ and $x_2(2)$, both of which reside in MEM 0. In stage 3 the inputs for the top PE are $x_3(0)$ and $x_3(1)$, both of which reside in MEM0.

Define the stage distance as the index delta of data feeding PEs in each stage. The stage distance for 16-point decimation in frequency FFT is 8 in stage 0, 4 in stage 1, 2 in stage 2 and 1 in stage 3. In general, the stage distance for stage i is equal to $N/2^{(i+1)}$. Memory conflicts occur when the stage distance falls in a single memory space. Since memory size is equal to $N/(2*M)$, memory conflicts occur in stage i if the following condition is satisfied:

$N/2^{(i+1)} < N/(2*M)$

$i > \log_2 (M)$

A stage i that satisfies condition ($i > \log_2 M$) will be referred to as a hazard stage; the rest of the stages are "safe" stages. For instance, in Figure 24, stage 2 and stage 3 are hazard stages. Define memory pair $(i, j)_t$ as memory location x(i) and x(j) for stage t. In stage 2, the following memory pairs are hazard pairs: $(0, 2)_2$, $(1, 3)_2$, $(4, 6)_2$, $(5, 7)_2$. Other pairs will be referred to as safe pairs, for instance $(0, 4)_1$. A pair $(i, j)_t$ is a hazard pair if t is a hazard stage.

To avoid conflicts, the proposed algorithm manipulates intermediate results in the memories using memory management operations. Let $x_i(t)$ and $x_j(t)$ be the i-th and j-th elements in stage t and i < j. Define the memory management operations as follows (see Figure 26):

1. **Normal Operation**: Input x_i and x_j are provided to the first and second inputs (a and b) of the PE. The results (c and d) are saved in x_i and x_j,
2. **Shuffle Operation:** Affects how PE results are written back in memory. In shuffle operation, the results (c and d) are saved in x_j and x_i,
3. **Swap Operation**: The swap operation affects the order of PE inputs. In swap operation, x_i is provided to b and x_j is provided to a,
4. **Swap and shuffle operation**: A PE operation can have both swap and shuffle memory operations at the same time.

If the algorithm detects a case when the input order is incorrect, the swap operation is performed.

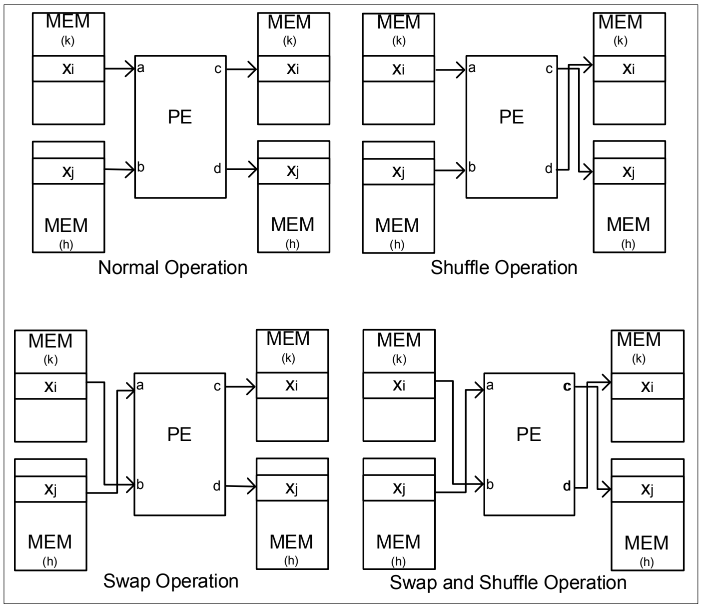


Figure 26. Memory Management Operations

The main idea of the memory management algorithm is to identify hazard pairs and perform memory management operations to resolve the hazard. Because data is rearranged in memory, the algorithm must track the data in the memories. One idea to track data is to use a separate memory to store the data indexes (i.e., pointers), as shown on Figure 27. This approach provides great flexibility in moving data in the memory. It also simplifies the reordering logic of the final stage hardware. There are two downsides to this approach. First, this technique increases the memory size. Since each memory location is tracked, the pointer memory size is N words. The width of the memory is $\log_2 N$. Second, it increases the time for loading PE operands. The operand loading consists of two cycles, loading the pointer from pointer memory and then loading the operands from operand memory. Another (less flexible) solution is to move data in memory in a methodical predictable way to simplify data tracking. Each operand is located based on a stage number and an operation number as demonstrated below. The only limitation is that the approach resolves hazards for next stage only. However, since it does not require extra memory and does not increase access time, this technique will be used in the below algorithm.

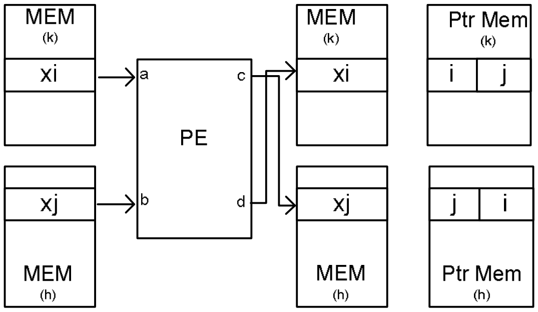


Figure 27. Tracking Shuffled Intermediate Results Using Pointers

The conflict-free algorithm is illustrated on Figure 28. On the left side is the outer loop of the algorithm executing stage by stage. On the right side, it shows the PE execution which can be summarized as follows:

- If the input data has incorrect order in the memory, the PE input is swapped,
- If the present output data pair will create a hazard in the next pipeline stage, the PE results are shuffled.

As a result of reordering data in the memories, results from the last stage should be reordered. Figure 29 shows the intermediate and final memory locations for memory conflict free 16-point FFT. Also shown in Figure 29 are the swap (in blue color) and shuffle operation (in Red color). Compare the following memory locations to those made in Figure 24:

- In Stage-2 the inputs for the top butterfly are $x_2(0)$ and $x_2(2)$. There is no memory conflict since $x_2(0)$ and $x_2(2)$ reside in MEM 0 and MEM 1, respectively.
- Similarly, in Stage-3 the inputs for the top butterfly are $x_3(0)$ and $x_3(1)$ which reside in MEM 0 and MEM 1, respectively.

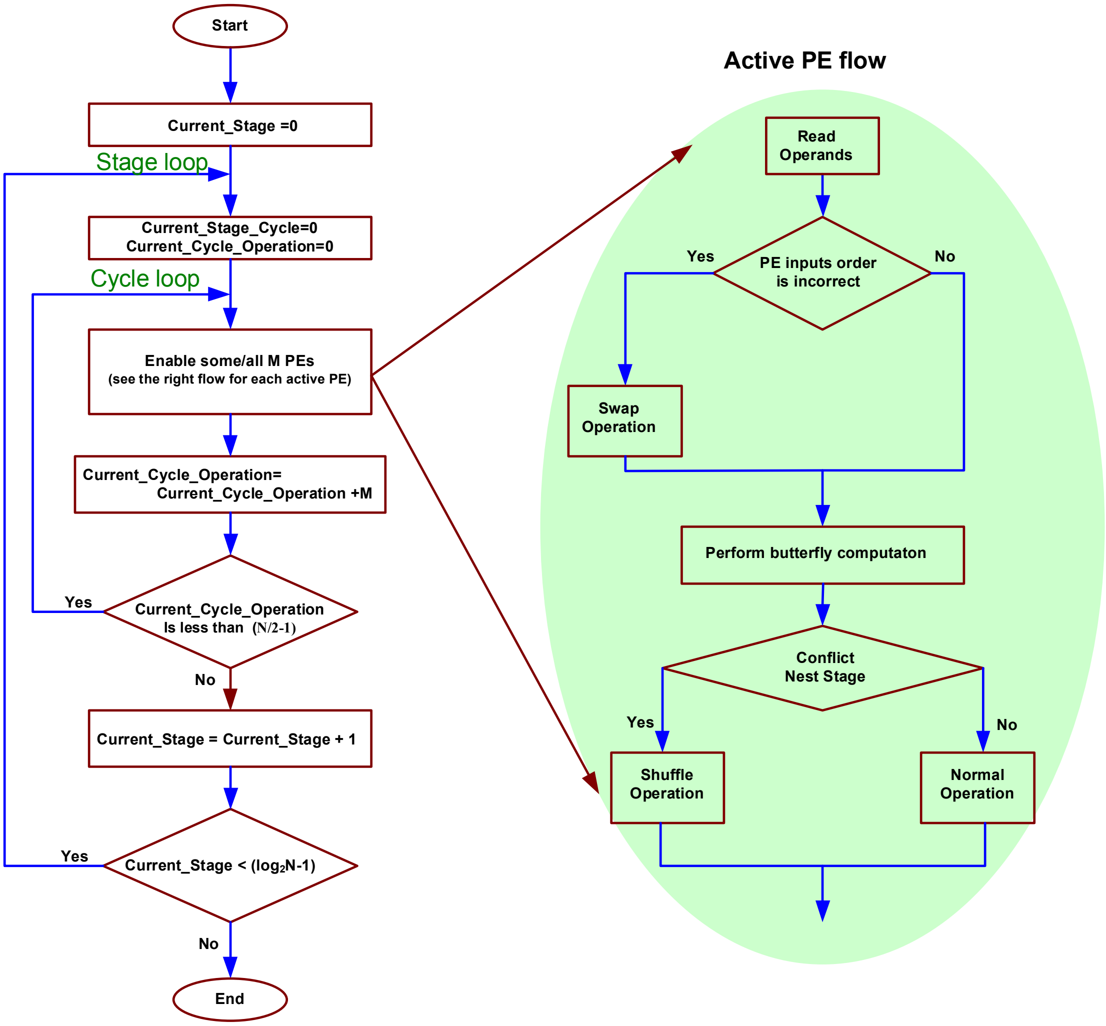


Figure 28. Memory Conflict Free Algorithm

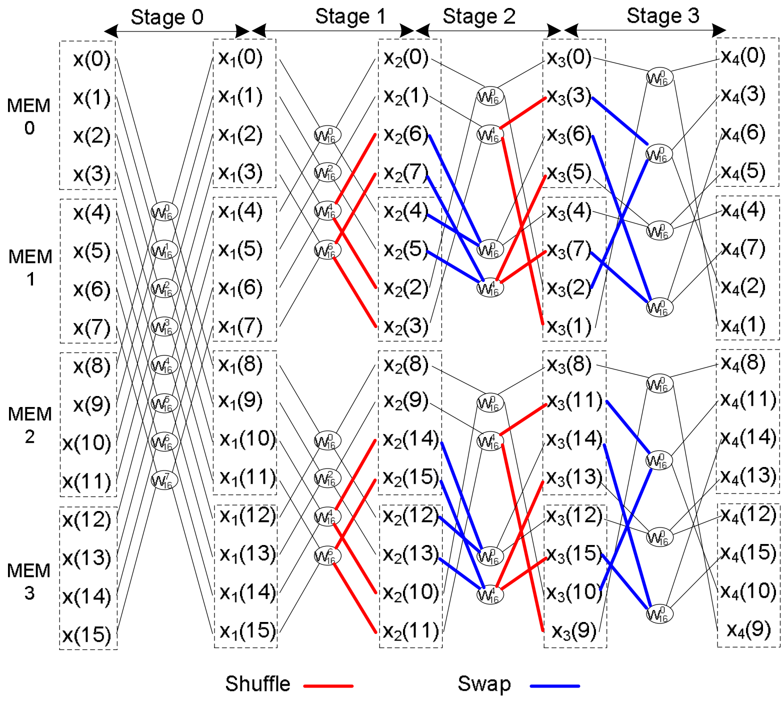


Figure 29. Memory Conflict Free 16-Point FFT

4.2.1. Algorithm Pseudocode

This section presents the pseudocode of the memory conflict free algorithm. To facilitate understanding the algorithm, Table 19 describes the main variables and counters used in the algorithm. The algorithm code (listed at the end of the section) starts by setting global variables (i.e., NUM_STAGES) and then executes the main loops. An iteration of the most inner loop executes the following steps:

1. Calculate operation indices,
2. Generate memory addresses and selects,
3. Read data from memory. Perform swap operation if needed,
4. Perform butterfly operation,
5. Perform shuffle operation if needed, and then write results to memory.

Table 19. The Main Variables and Counters Used in the Algorithm

Variable	Description
NUM_STAGES	Total Number of Stages, which equal to $\log_2 N$
SAFE_STAGE	The highest stage which is considered a safe stage. It is expressed as follows: SAFE_STAGE = $\log_2$ PE_NUM
Cycles_Per_Stage	Number of cycles per stage, which is expressed as: Cycles_Per_Stage= N/(2*PE_NUM)
Mem_size	Memory Size = N/(2*PE_NUM)
Stage_Counter	The stage counter”
Cycle_Counter	The cycle counter
PE_ID	The PE_ID Counter
PE_Cycle_Counter	An intermediate counter, which is used to generated memory addresses and selects. It is constructed as follows:: PE_Cycle_Counter =[PE_ID, Cycle_Counter]
Current_Group	An intermediate counter used to generate RAM selects. It is constructed as follows: Current_Group = the upper k-bits of PE_Cycle_Counter where k= Stage_Counter
Current_Operation	An intermediate counter used to generate ROM selects. Current_Group = the lower k-bits of PE_Cycle_Counter where k= NUM_STAGES -Stage_Counter -1

Algorithm Pseudocode

```
// Preparation Step
Number_Of_Stages   = $\log_2$NUMBER_OF_FFT_POINTS
Cycles_Per_Stage  = N/(2*NUMBER_OF_PE)
Memory_Size       = N/$2^{(NUMBER\_OF\_PE+1)}$
Safe_Stage        = $\log_2$NUMBER_OF_PE
// Start main loops
for Stage_Counter=0 to (Number_Of_Stages -1)
 Group_Size = N/$2^{(Stage\_Counter+1)}$
 for Cycle_Counter=0 to (Cycles_Per_Stage -1)
  for PE_ID=0 to (NUMBER_OF_PE -1)
```

```
// Calculate Operation Indices
PE_Cycle_Counter = Cycles_Per_Stage *
                      PE_ID
                      + Cycle_Counter
Stage_Counter_Rev = Number_O_Stages - Stage_Counter - 1
Current_Group     = floor(PE_Cycle_Counter/
                          $2^{Stage\_Counter\_Rev}$)
Current_Operation = PE_Cycle_Counter mod $2^{Stage\_Counter\_Rev}$

// Generate memory selects and addresses
// The following notation means accessing address M0_address
// in "M0_select" segment of memory: $Memory_{(M0\_Select)}$ [ M0_addr]
// Memory assumed one linear array for matlab simulations.
M0_addr = Cycle_Counter
If Stage_Counter <= Safe_Stage
  M1_addr = M0_addr
Else
  K = Stage_Counter - Safe_Stage
  M1_Addr = Invert upper K-bits of M0_Addr0
End

If Stage_Counter <= Safe_Stage
  Group_Offset = Current_Group * N /$2^{Stage\_Counter}$
  Group_Count  = PE_Cycle_Counter mod Group_Size
  Memory_Count = floor (Group_Count / Memory_Size)
  Offset       = Memory_Count * Memory_Size
  M0_Select    = Offset + Group_Offset
  M1_Select    = Offset + Group_Offset + Group_Size
Else
  Memory_Count = PI_ID
  Offset     = 2 * Memory_Count * Memory_Size
  M0_Select = Offset;
  M1_Select = Offset + 2 * Memory_SiZe
End

// Read data and perform swap operation if needed
If Stage_Counter > SAFE_STAGE
   AND Cycle_Counter [ NUM_SAGES - Stage_Counter -1] ==1
  // Read with Swap
  M1_data = $Memory_{(M0\_Select)}$ [ M0_addr ]
  M0_data = $Memory_{(M1\_Select)}$ [ M1_addr ]
else
```

```
     // Read with no swap
     M0_data = $Memory_{(M0\_Select)}$ [ M0_addr ]
     M1_data = $Memory_{(M1\_Select)}$ [ M1_addr ]
   End

   // Read Twiddle
   ROM_SELECT   = PE_ID
   ROM_Address = Current_Operation * $2^{Stage\_Counter}$
    W    = $ROM_{(Stage\_Counter,\ ROM\_SELECT)}$ [ROM_Address ]

   // Enable PE to perform FFT butterfly operation
   [Result1, Result0] =
         $PE_{PE\_ID}$(M0_data, M1_data, W);

   // Write the results to the Memory
   // Perform the shuffle operation if required
   If  Stage_Counter >= Safe_Stage
       AND Stage_Counter < (Number_Of_Stages -1)
       AND Cycle_Counter [ NUM_SAGES - Stage_Counter -2] ==1
     // Shuffle the results
     $Memory_{(M0\_Select)}$ [ M0_addr ] = Result1
     $Memory_{(M1\_Select)}$ [ M1_addr ] = Result0
   Else
     // No Shuffling
     $Memory_{(M0\_Select)}$ [ M0_addr ] = Result0
     $Memory_{(M1\_Select)}$ [ M1_addr ] = Result1
   End

  end // PE_ID
 end // Cycle_Counter loop
end // Current_Stage loop
```

4.3 64-point Design

This section illustrates in detail the PE operand manipulation and memory contents for a 64-point DIF FFT using four PEs. Since M=4, therefore:

- There are eight memories and each has a size of eight words.
- There are four ROMs, each with size of eight words,
- There are eight cycles per stage.

The following tables detail the operation of the PEs and illustrate the memory contents through the stages. Table 20 gives the PE operand pairs for Stage 0. The rows give the operand pairs for PE_0, PE_1, PE_2 and PE_3. The columns give the pairs for each of the eight cycles of Stage 0. For example, at cycle 0:

- PE_0 input operands will be MEM[0] and MEM[32],
- PE_1 input operands will be MEM[8] and MEM[40],
- PE_2 input operands will be MEM[16] and MEM[48],
- PE_3 input operands will be MEM[24] and MEM[56].

Table 21-Table 25 show the PE operand pairs for Stages 1-5. Underlined pairs indicate a shuffle operation and blue-colored pairs indicate a swap operation. Since Stages 0-2 are safe stages, the first shuffle operation starts in Stage 2 to prevent hazards in Stage 3. Table 26-Table 33 list the memory contents across the stages. For example, the output of Stage 2 has the memory contents for Memory 0 as follows: 0, 1, 2, 3, 12, 13, 14, and 15.

Table 20. Stage-0 PE Operand Pairs

PE	**Stage-0 Cycles**							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0,32	1,33	2,34	3,35	4,36	5,37	6,38	7,38
1	8,40	9,41	10,42	11,43	12,44	13,45	14,46	15,47
2	16,48	17,49	18,50	19,51	20,52	21,53	22,54	23,55
3	24,56	25,57	26,58	27,59	28,60	29,61	30,61	31,63

Table 21. Stage-1 PE Operand Pairs

PE	Stage-1 Cycles							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0,16	1,17	2,18	3,19	4,20	5,21	6,22	7,23
1	8,24	9,25	10,26	11,27	12,28	13,29	14,30	15,31
2	32,48	33,49	34,50	35,51	36,52	37,53	38,54	39,55
3	40,56	41,57	42,58	43,59	44,60	45,61	46,62	47,63

Table 22. Stage-2 PE Operand Pairs

PE	Stage-2 Cycles							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0,8	1,9	2,10	3,11	4,12	5,13	6,14	7,15
1	16,24	17,25	18,26	19,27	20,28	21,29	22,30	23,31
2	32,40	33,41	34,42	35,42	36,44	37,45	38,46	39,47
3	48,56	49,57	50,58	51,59	52,60	53,61	54,62	55,63

Table 23. Stage-3 PE Operand Pairs

PE	Stage-3 Cycles							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0,4	1,5	2,6	3,7	12,8	13,9	14,10	15,11
1	16,20	17,21	18,22	19,23	28,24	29,25	30,26	31,27
2	32,36	33,37	34,38	35,39	44,40	45,41	46,42	47,43
3	48,52	49,53	50,54	51,55	60,56	61,57	62,58	63,59

Table 24. Stage-4 PE Operand Pairs

PE	Stage-4 Cycles							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0,2	1,3	6,4	7,5	12,14	13,15	10,8	11,9
1	16,18	17,19	22,20	23,21	28,30	29,31	26,2	27,25
2	32,34	33,35	38,36	39,37	44,46	45,47	42,40	43,41
3	48,50	49,51	54,52	55,53	60,62	61,63	58,56	59,57

Table 25. Stage-5 PE Operand Pairs

PE	Stage-5 Cycles							
	0	1	2	3	4	5	6	7
0	0,1	3,2	6,7	5,4	12,13	15,14	10,11	9,8
1	16,17	19,18	22,23	21,20	28,29	31,30	26,27	25,25
2	32,33	35,34	38,39	37,36	44,45	47,46	42,43	41,40
3	48,49	51,50	54,55	53,52	60,61	63,62	58,59	57,56

Table 26. Memory-0 Contents Across FFT Stages

Address	Stages						
	Input	0	1	2	3	4	5
0	0	0	0	0	0	0	0
1	1	1	1	1	1	3	3
2	2	2	2	2	6	6	6
3	3	3	3	3	7	5	5
4	4	4	4	12	12	12	12
5	5	5	5	13	13	15	15
6	6	6	6	14	10	10	10
7	7	7	7	15	11	9	9

Table 27. Memory-1 Contents Across FFT Stages

Address	Stages						
	Input	0	1	2	3	4	5
0	8	8	8	8	8	8	8
1	9	9	9	9	9	11	11
2	10	10	10	10	14	14	14
3	11	11	11	11	15	13	13
4	12	12	12	4	4	4	4
5	13	13	13	5	5	7	7
6	14	14	14	6	2	2	2
7	15	15	15	7	3	1	1

Table 28. Memory-2 Contents Across FFT Stages

Address	**Stages**						
	Input	**0**	**1**	**2**	**3**	**4**	**5**
0	16	16	16	16	16	16	16
1	17	17	17	17	17	19	19
2	18	18	18	18	22	22	22
3	19	19	19	19	23	21	21
4	20	20	20	28	28	28	28
5	21	21	21	29	29	31	31
6	22	22	22	30	26	26	26
7	23	23	23	31	27	25	25

Table 29. Memory-3 Contents Across FFT Stages

Address	**Stages**						
	Input	**0**	**1**	**2**	**3**	**4**	**5**
0	24	24	24	24	24	24	24
1	25	25	25	25	25	27	27
2	26	26	26	26	30	30	30
3	27	27	27	27	31	29	29
4	28	28	28	20	20	20	20
5	29	29	29	21	21	23	23
6	30	30	30	22	18	18	18
7	31	31	31	23	19	17	17

Table 30. Memory-4 Contents Across FFT Stages

Address	**Stages**						
	Input	**0**	**1**	**2**	**3**	**4**	**5**
0	32	32	32	32	32	32	32
1	33	33	33	33	33	35	35
2	34	34	34	34	38	38	38
3	35	35	35	35	35	37	37
4	36	36	36	44	44	44	44
5	37	37	37	45	45	47	47
6	38	38	38	46	42	42	42
7	39	39	39	47	43	41	41

Table 31. Memory-5 Contents Across FFT Stages

Address	Stages						
	Input	0	1	2	3	4	5
0	40	40	40	40	40	40	40
1	41	41	41	41	41	43	43
2	42	42	42	42	46	46	46
3	43	43	43	43	47	45	45
4	44	44	44	36	36	36	36
5	45	45	45	37	37	39	39
6	46	46	46	38	34	34	34
7	47	47	47	39	35	33	33

Table 32. Memory-6 Contents Across FFT Stages

Address	Stages						
	Input	0	1	2	3	4	5
0	48	48	48	48	48	48	48
1	49	49	49	49	49	51	51
2	50	50	50	50	54	54	54
3	51	51	51	51	55	53	53
4	52	52	52	60	60	60	60
5	53	53	53	61	61	63	63
6	54	54	54	62	58	58	58
7	55	55	55	63	59	57	57

Table 33. Memory-7 Contents Across FFT Stages

Address	Stages						
	Input	0	1	2	3	4	5
0	56	56	56	56	56	56	56
1	57	57	57	57	57	59	59
2	58	58	58	58	62	62	62
3	59	59	59	59	63	61	61
4	60	60	60	52	52	52	52
5	61	61	61	53	53	55	55
6	62	62	62	54	50	50	50
7	63	63	63	55	51	49	49

4.4 Dynamic Range and Signal to Noise Ratio

This section discusses the impact of data representation on dynamic range and signal to noise ratio. For a signed integer with a binary word length of b+1, the dynamic range in decibels is expressed as [28]:

$$\begin{aligned}\text{Dnamic Range}_{\text{dB}} &= 20\times\log_{10}\left(\frac{\text{Largest Possible Word Value}}{\text{Smallest Possible Word Value}}\right)\\ &= 20\times\log_{10}\left(\frac{2^b-1}{1}\right)\\ &= 20\times\log_{10}\left(2^b-1\right)\\ &\approx 20\times\log_{10}\left(2^b\right)\ dB\\ &= 6.02\times b\end{aligned}$$

The above equation implies that the dynamic range in decibels is proportional to the world length. Table 34 illustrates the dynamic range for several fixed-point representations. Clearly, the 32-bit representation provides a much better dynamic range compared to 16-bit and 24-bit representation. Further, the 32-bit dynamic range is adequate for cell phone and communication applications.

Table 34. Dynamic Rage for Various Fixed-Point Representations

(b+1)	Dynamic Rage (dB)
4	18.1
8	42.1
12	66.2
16	90.3
24	138.5
32	186.6

At each stage of the FFT processing, errors are introduced due to round-off and truncation. The error accumulates and increases for every successive stage. Reference [29] analyzed and derived the aggregate error at the output stage. The analysis was done for two cases in which the overflow condition does not occur at any stage. While FFT implementations manage overflow in different methods, the below analysis provides a first order relationship between SNR and word length.

The first case requires the inputs to be:

$$|x[n]| < \frac{1}{N}$$

In this case, the signal to noise ratio at the FFT output is expressed as [29]:

$$SNR_{Output\ \ Stage} = \frac{2^{2b}}{N^2}$$

$$SNR_{Output\ \ Stage}(dB) = 6.02 \times b - 20 \times \log_{10} N$$

Hence, the SNR_{output_stage} is inversely proportional to N^2. Moreover, if N is doubled, the word length must increase by 1-bit to maintain the same SNR_{output_stage}.

The second case relaxes the conditions on the inputs:

- $|x[n]| < 1$,
- Requires attenuation of ½ at the input of each butterfly.

In this case, the signal to noise ratio at the FFT output is expressed as [29]:

$$SNR_{Output\ \ Stage} = \frac{2^{2b}}{4N}$$

$$SNR_{Output\ \ Stage}(dB) = 6.02 \times b - 10 \times \log_{10} N - 0.6$$

Hence, the SNR_{output_stage} is inversely proportional to N rather than N^2. Moreover, if N is doubled, the word length must increase by ½ bit to maintain the same SNR_{output_stage}.

Table 35 illustrates the SNR_{output_stage} for word lengths of 16-bits and 32-bits for both cases. The table shows the SNR results for different values of *N*. For 16-bit word lengths, the SNR can be severely degraded when N=4096 depending on the overflow mechanism. Also, the table shows the SNR is dependent on how the overflow condition is handled.

Table 35. SNR for Various Fixed-Point Representations

N	Case 1		Case 2	
	SNR_{output_stage} (dB) word length =16	**SNR_{output_stage} (dB) word length =32**	**SNR_{output_stage} (dB) word length =16**	**SNR_{output_stage} (dB) word length =32**
64	60.2	156.5	77.7	174.0
256	48.2	144.5	71.6	168.0
1024	36.1	132.4	65.6	161.9
4096	24.1	120.4	59.6	155.9

In summary, 32-bit word length offers superior dynamic range and SNR, independent of how the overflow condition is handled. This word size will be used in the processor implementation in the following chapters.

4.5 Summary

This chapter discussed the two forms of the switch-based architecture: pipelined and non-pipelined. Next, the memory conflict free algorithm was presented. Lastly, the data format impact on the dynamic range was examined.

Chapter 5

Logic Design

This chapter (and the next two chapters) presents the implementation details of a switch-based FFT processor. The motivation for the implementation is to demonstrate advantages of the switch-based architecture specifically in the areas of performance, configurability and power-scalability. Furthermore, the switch design timing and power overhead will be analyzed to answer the following questions. How much power is dissipated in the switch? And, how much of the critical path is in the switch circuit?

This chapter focuses on the RTL design. It starts by discussing the key design specifications and the design overview. Next, the functional blocks and the operations are presented in detail. Finally, the design complexity is analyzed.

5.1 Design Specifications

Table 36 presents the design specifications of the implementation. Since the data are complex, they are represented by a double word (i.e., 64-bit). The most significant 32-bits are the imaginary component and least significant 32-bits are the real component of the data. Hence, data path components (i.e., adders and multipliers) should support 32-bit word sizes. Moreover, the RAM entry stores a double word (i.e., 64-bit).

Table 36. Design Specifications

Parameter	**Specification**
FFT Type	Radix-2 DIF
Architecture	Switch-based, non-pipelined
Number of points (*N*)	Configurable to process: 64, 256, 1024 and 4096 points
Number of blocks (*B*)	Configurable to process up to 15 blocks of *N*-points, one block is *N*-points
Number of PEs (*M*)	Four
Word length	Word length is 32-bit.
Data format	Data is represented in: • Two's complement, • Fixed point format, with 16 fractional bits and 16 integer bits.
Memory Arrays	Two sets of memories, each set has 8 memories; RAM size is 512 entries (i.e., double word)
Power scalability	Power scales linearly with number of active units PEs
Target Clock Rate	200 MHz
Target Throughput (N=1024)	53 MSPS
Target Technology	Low power 65-nm CMOS bulk substrate [26]

5.2 Design Overview

Figure 30 illustrates an overview of the processor design. The design consists mainly of the memory sub-system and the switch sub-system. The memory sub-system contains two sets of RAMs and an external memory controller. The primary task of the memory sub-system is to move data between RAMs and the external memory with minimal interruption to FFT processing. The RAMs serve as caches in the design. When an FFT computation is being preformed on one set of RAMs, the other set is scheduled for data movement with external memory. Each RAM set employs eight 512x64-bit RAM. Each RAM entry contains the 32-bit real component and 32-bit imaginary component of the complex data.

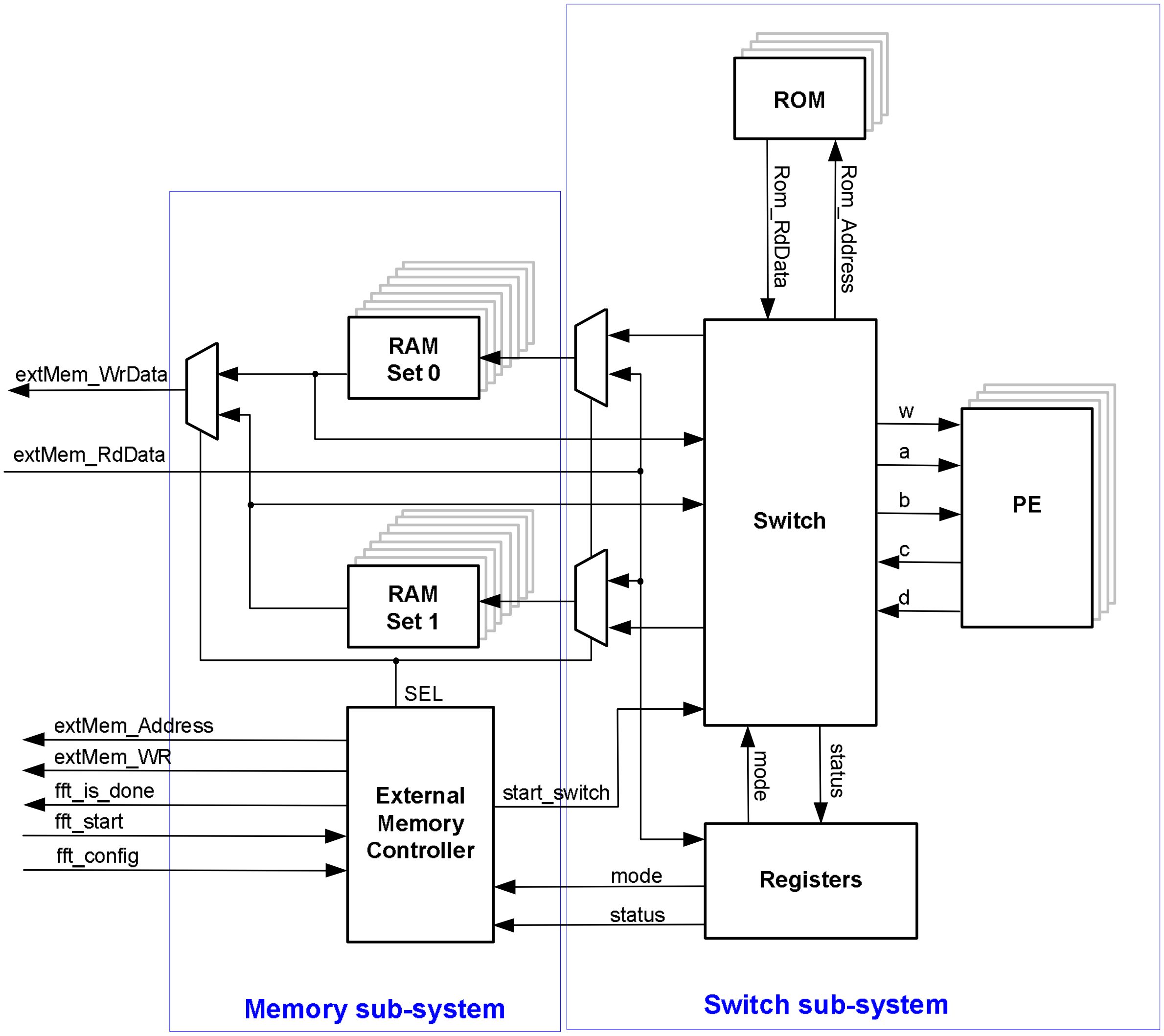


Figure 30. Design Overview

The switch sub-system consists of the switch fabric, four PEs, four ROMs and a registers unit. The switch sub-system computes the FFT for one of the RAM sets. Once the start signal from external memory controller is received, the switch sub-system computes the FFT and then indicates the completion status to the registers unit.

The processor can be programmed to process multiple blocks of N-points. A block consists of one set of N-points. Once configured, the processor computes the FFT for an entire block without interruptions. For example, if the processor is programmed to process thirty blocks of 64 points, then it computes thirty sets of 64-points and then transitions to an idle state. Additionally, the block source and block destination addresses are programmable.

When configured to execute more than one block, the processor assigns the even blocks to set_0 and odd blocks to set_1. A 5-block execution example is illustrated in Figure 31. At t_0, block 0 is loaded in RAM set_0. Then at t_1, block 0 is processed in the switch subsystem. Concurrently, block 1 is loaded into RAM set_1. At t_2, block 0 is written back to external memory, and then block 2 is loaded into RAM set_0. At the same time, block 1 FFT computation is performed. The timing diagram shows that the switch is idle during the loading of the first block and writing of the last block. Since the loading time is less than the FFT processing time, the idle time can be ignored when the number of blocks is more than five blocks, i.e.,

$$T_7 - T_0 \approx 5 \times T_{OneBlock}$$

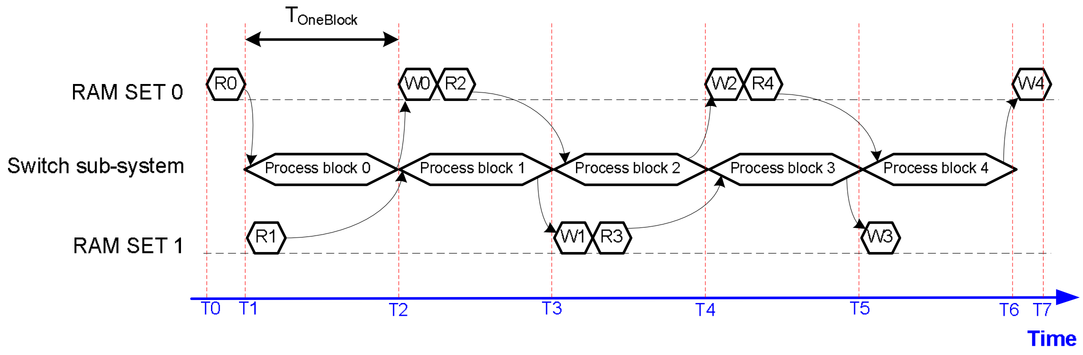


Figure 31. Block Processing Timing Diagram

5.3 Functional Block Descriptions

This section examines the functional blocks in the memory sub-system and switch sub-system.

5.3.1. Memory Sub-System

The memory subsystem is responsible for moving RAM data and synchronizing activities with the switch and the outside system. After it completes loading the first block, the external memory controller asserts the “start_switch” signal (see Figure 30), instructing the switch to start processing. The controller monitors the status of the block processing. Each completed block is written back to the external memory. When the last block is written, the controller asserts the output signal, “fft_done”, as an indication to the outside system. The memory subsystem consists of two sets of RAMs and the controller. The remainder of this section examines them in detail.

The RAMs

There are two RAM sets, as shown in Figure 30. Each set has eight RAMs. The size of the RAM is determined by the largest number of points processed (i.e., 4096-points) and the number of PEs:

RAM entries = 4096/(2* Number_of_PEs) = 512 entries

Each RAM entry contains a 32-bit real and a 32-bit imaginary component of the complex data. Hence, the entry size is 64-bits. Therefore, the RAM size is 512x64-bits. The RAM signal interface is defined in Table 37. There are three modes of operation: Read access, Write access and No-access. The RAMs are implemented using a typical 65nm CMOS SRAM. The timing and power specifications of RAM cells are shown in Table 38 and Table 39. The RAM specifications were estimated based on the guidelines presented in [27]. The RAM latches the input data internally, and as a result, the data and address are required to meet the typical 65nm FF setup time. Also, the output signals are driven off internal RAM FFs, and therefore the output will be available after the typical 65nm FF CLK→Q delay. RAM design is modeled with a library file which captures the specifications in Table 38 and Table 39.

Table 37. RAM Interface

Signal Name	Input/Output	Size	Description
clk	Input	1	Clock signal
Address	Input	9	Address bus
SEL	Input	1	RAM is accessed when SEL=1 When RAM is not accessed (SEL=0), the previous RdData bus is held
WR	Input	1	Access type: READ (WR=0) or WRITE (WR=1)
WrData	Input	64	Write data bus
RdData	Output	64	Read data bus

Table 38. RAM Timing Specification

Signal Name	Required Time	Available Time
clk	Clock	
Address	300ps before rise clk	
CS	100ps before rise clk	
WR	0ps before rise clk	
WrData	100ps before the rise Similar to FF setup	
RdData		300ps after the clock rise (Clk →FF_q)

Table 39. Energy/Power Specification

Operation	Energy
No Access (Leakage)	30 µW
READ Energy	25 pJ/access
WRITE Energy	30 pJ/access

External Memory Controller

The external memory controller serves two main purposes. It manages the data movement between the RAMs and the rest of the system. In addition, it performs address mapping between RAMs and the external memory. The state machine of the controller is shown in Figure 32.

Initially, the state machine is in the *IDLE* state, in which no activities are conducted to minimize dynamic power. The outside system informs the processor to start FFT computation by asserting the "fft_start" signal. As a result, the state machine transitions to the *RD* $Block_0$ state. In this state, the controller generates the addresses and the access signals to read the first block into set_0 RAMs. Once first block is loaded, the next state depends on the number of blocks. If there is only one block, the state machine waits till the block is processed (in *WAIT for* $Block_i$ state). It then writes the results to external memory (in *WR Last Block* state) and transitions to the idle state.

If the number of blocks is greater than one, the next state (after *RD* $Block_0$ state) is *RD* $Block_{i+1}$. Then the state machine loops through three states per $block_i$. Basically, while $block_i$ is being processed, the machine writes the results of $block_{i-1}$ (in *WR* $Block_{i-1}$ state), reads the data of $block_{i+1}$ (in *RD* $Block_{i+1}$ state) and then waits for $block_i$(in *WAIT for* $Block_i$ state). If the current block is the last block then

- The *RD* $Block_{i+1}$ state is skipped,
- After *WAIT for* $Block_i$ state, the machine transitions to *WR Last Block* state.

Following writing the last block, the *WR Last Block* state asserts the signal "fft_done", as an indication to the outside system that the processing has completed. Then outside system can start programming the processor to compute FFT for another set of blocks.

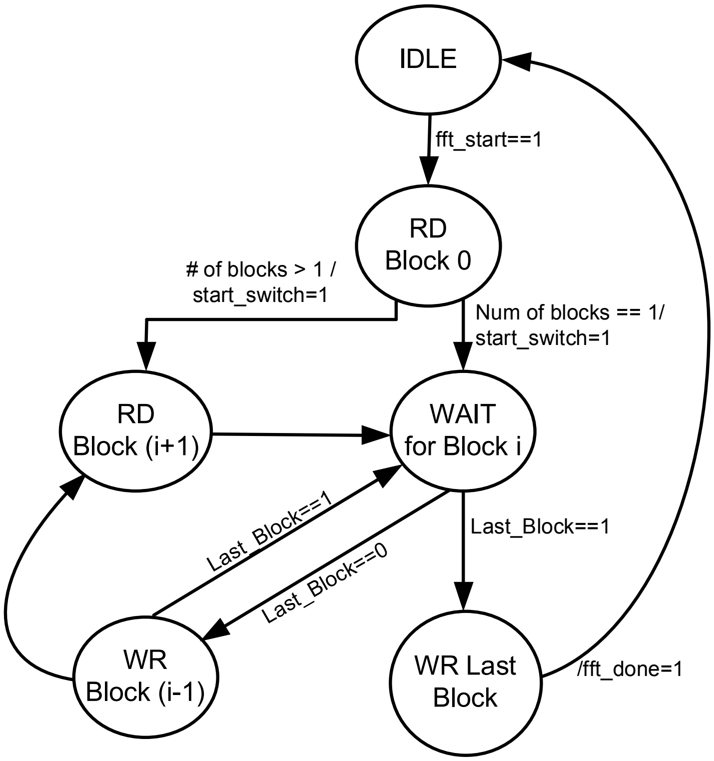


Figure 32. External Memory Controller State machine

The data and addresses are directed throughout the design using 2-1 muxes controlled by the *set_0_sel* signal, as shown in Figure 33. The controller asserts *set_0_sel* during reading an even block from external memory (e.g., block 0). When the controller asserts *set_0_sel*:

- The data flows between the external memory controller and the set_0 RAMs. The RAM addresses are generated by the controller.
- The data flows between set_1 RAM and the switch. The set_1 RAM addresses are generated by the switch.

To read the next block, the controller de-asserts set_0_sel, allowing data to flow between RAM set_1 and external memory.

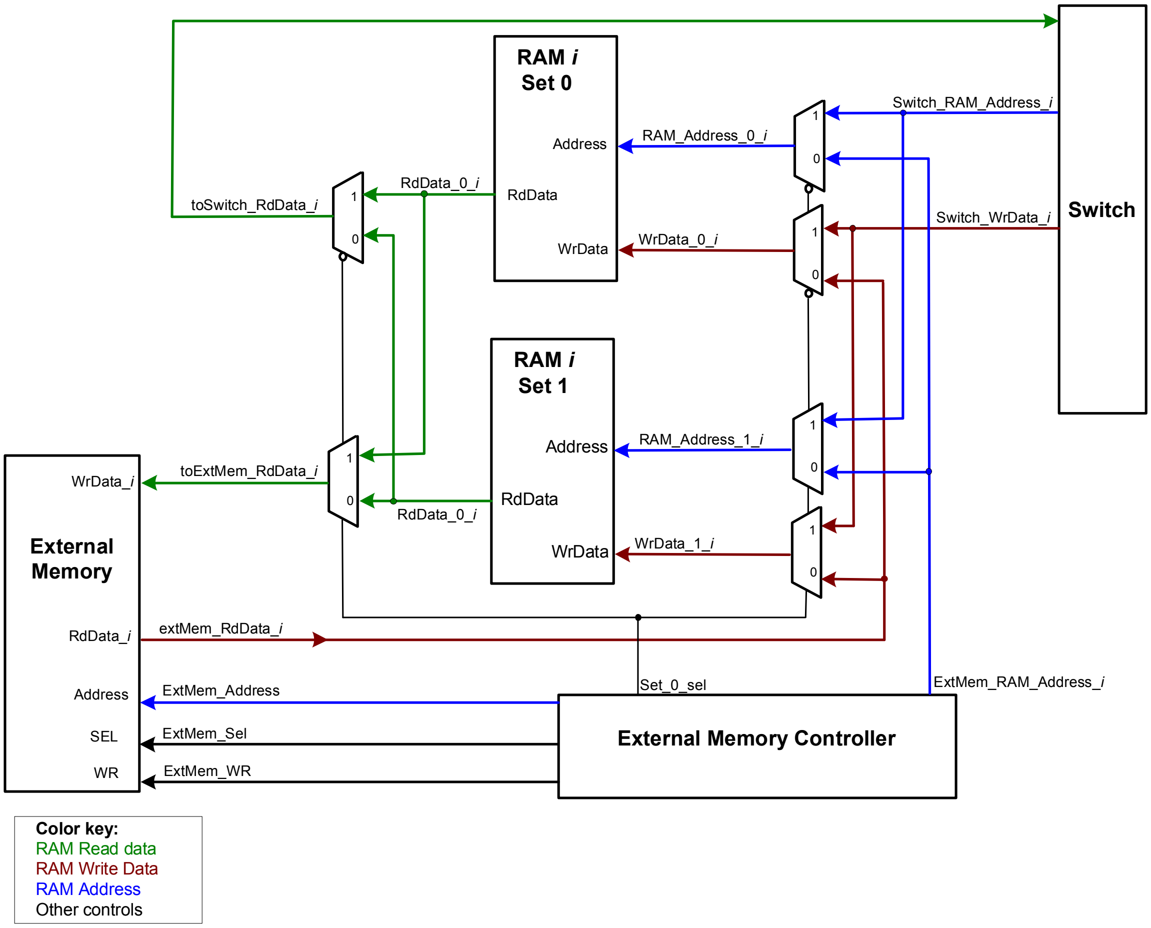


Figure 33. Routing address and data in the design

The external memory address has three portions as shown in Figure 34: The upper 12 bits of the address is programmed during the processor configuration. More details are in the register unit section. The *block number* is initially set to zero and then is incremented sequentially by the controller until all blocks are processed. The maximum value for *block number* is 31. The *block offset* is the index of the 64-bit data in the *N*-point block.

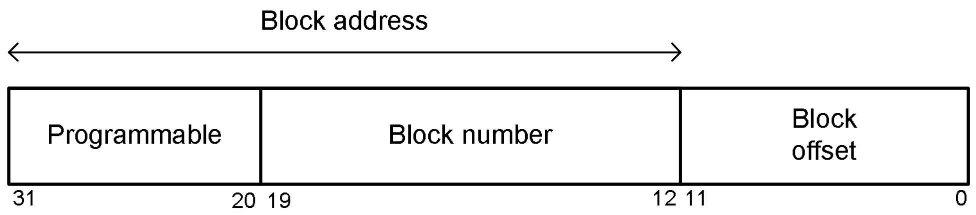


Figure 34. The Three Portions of External Memory Address

The controller performs address mapping between RAM and external memory. During read access, the external data is mapped to the *i*-th address of RAM_j, where *i*=extMem_Address[9:0] and *j*= extMem_Address[11:10]. However, during writing data to the external memory, the mapping is more complex since the algorithm changes the result order. The controller reorders the data so that the data matches the output of decimation-in-frequency radix-2 FFT algorithm. This mapping is outlined as follows. To map an address RAM_Adress of RAM*j* to extMem_Address:

extMem_Address[0] = RAM_Address[0]

extMem_Address[1] = RAM_Address[1] xor RAM_Address[0]

extMem_Address[2] = RAM_Address[2] xor RAM_Address[1]

…..

extMem_Address[i] = RAM_Address[i] xor … xor RAM_Address[i-1]

extMem_Address[11:10] = *j*

5.3.2. The Switch Sub-System

The switch subsystem is composed of the switch fabric, four PEs, four ROMs and the register unit. The remainder of this section examines the blocks in detail.

The Switch Fabric

The main function of switch fabric is to direct data from the RAMs and ROMs to PEs, and then it directs the results back to the RAMs. It also generates the RAM and

ROM addresses for the accessed data. The main components of the switch are: the data routing MUXes, the state machine and the address generation logic.

The switch MUXes are illustrated in the exploded view of the switch in Figure 35. Also shown in the figure is the connectivity of the switch to the rest of the design. The switch exchanges control and status data with external memory controller and the registers block. All data buses are 64-bit.

The MUXes select the appropriate input data for the PE ports **a**, **b** and **w**. The inputs for the MUXes which feed **a** and **b** ports are the RAM read data. The inputs for MUXes which feed the **w** port are the ROM read data. Additionally, the inputs for the RAM write data MUX are the PE results (i.e., data from the **c** and **d** ports). The MUX selects are generated by the state machine.

An exploded view of the switch state machine is shown in Figure 36. Initially, the machine is the IDLE state where, all the logic is inactive to reduce dynamic power. After loading the first block in set_0, the external memory controller asserts the start_switch signal, instructing the switch to start FFT computations. The switch loops through the following states until the block is completed: *Read RAMs*, *PE COMP* and *Write RAMs*. During the *Read RAMs* state, the switch reads the data from RAMs and ROMs. The PEs performs the butterfly operations during the *PE COMP* state. In the *Write RAMs* state, the PE results are written back to the RAMs. When all blocks are completed, the *Write RAMs* state transitions to the *IDLE* state. If there are more blocks to process, the next state is the *Next FFT Block* state. The *Next FFT Block* state prepares the state machine for the next block by resetting the pointers and counters. Recall that N-point processing has $\log_2 N$ stages; each stage contains $N/2$ PE operations. Since the design employs four PEs and each stage requires $N/8$ cycles, each cycle performs four butterflies. Once it is configured, the design computes the number of cycles. Each cycle consists of three (or more) micro-cycles (i.e., clocks):

- RAM read access micro-cycle,
- PE micro-cycle,

- RAM write micro-cycle,
- Waiting cycles if one PE (or more) is inactive (i.e., not available)

The operations in one micro-cycle must complete within one system clock. Figure 37 illustrates the relationship of blocks, stages, cycles and clocks.

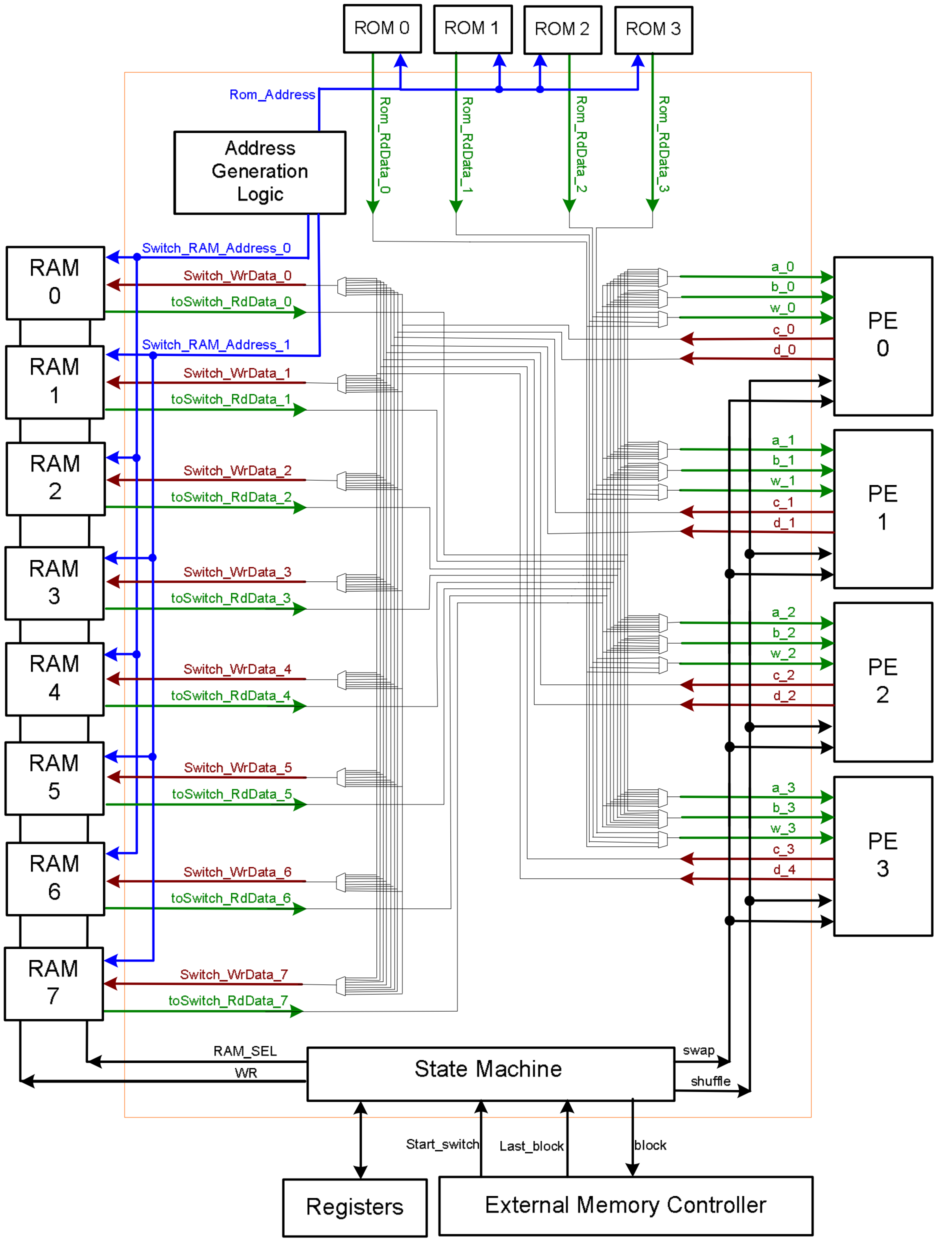


Figure 35. Switch Connectivity with the Design Components

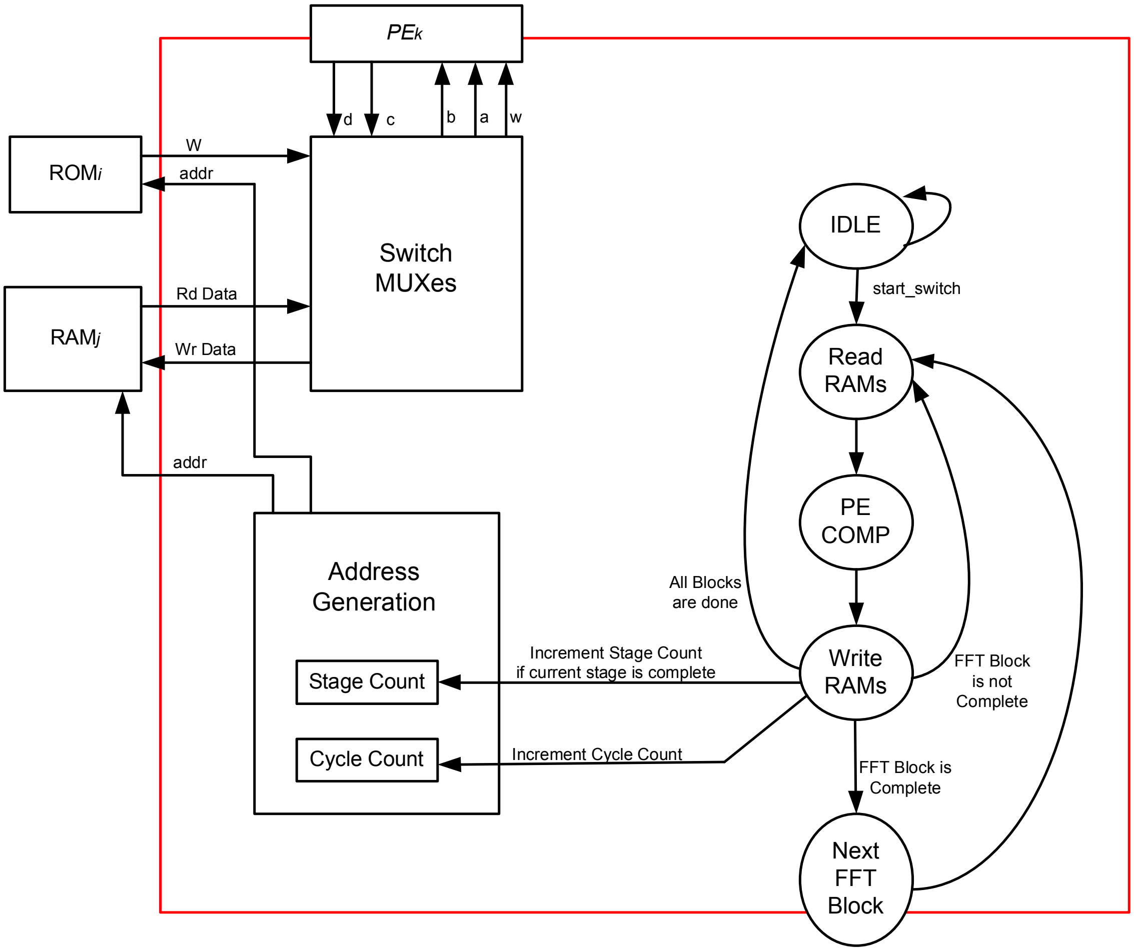


Figure 36. Switch State Machine

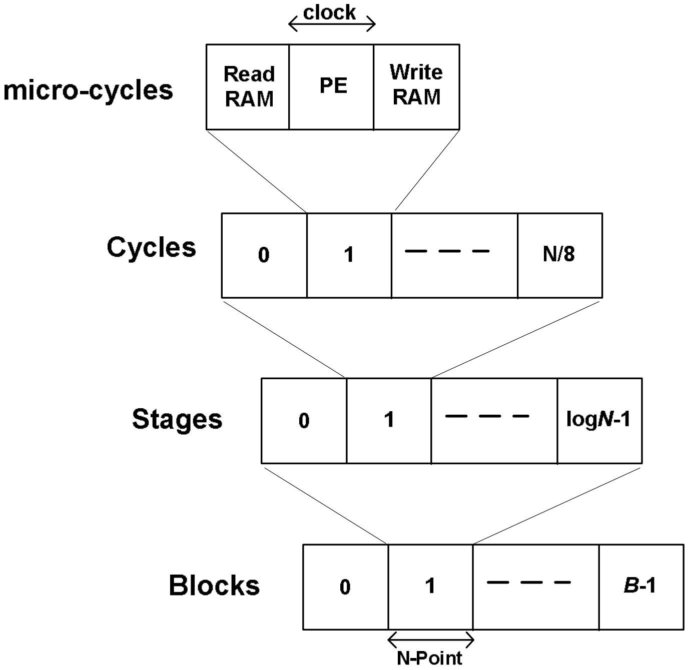


Figure 37. Blocks, Stages, Cycles and Clock Relationship

The address generation logic employs a cycle counter and a stage counter to compute the addresses for the RAMs and ROMs (see Figure 36). The counters are initialized to zero in the *IDLE* state and *Next FFT Block* state. The cycle counter is incremented in the *Write RAMs* state. The stage counter is incremented in the *Write RAMs* if a stage has completed.

The ROM address generation is illustrated in Figure 38. The address is generated by applying left-shift operation on the cycle counter. The shift count is expressed as

Shift Count = $\text{Log}_2 4096 - \text{Log}_2 N + \text{Stage\_Counter}$

where N is 64, 256, 1024 or 4096. Also, $\text{Log}_2 4096 = 12$.

The "$\text{Log}_2 4096 - \text{Log}_2 N$" in shift count is required for address adjustment. The reason for this adjustment is because all configurations (i.e., 64-point, 256-point, 1024-point or 4096-point) share the same ROM which is based on W_{4096}. As a result, the ROM addresses of other configurations (i.e., 64-point, 256-point and 1024-point) require adjustment by a factor of 4096/*N*, as illustrated in the following example:

$$W_{64}^{3} = W_{64*64}^{3*64} = W_{4096}^{192}$$
$$W_{256}^{12} = W_{256*16}^{12*16} = W_{4096}^{192}$$
$$W_{1024}^{48} = W_{1024*4}^{48*4} = W_{4096}^{192}$$

The Verilog code for ROM address generation is illustrated in Figure 39. The code is configured to compute the ROM address based on the programmed mode (i.e., the value of *N*.) The ROM address is generated by concatenating cycle_q and set of zeros. Then a right shift of the amount ($\log_2 N$-1 – stage_q) is performed. Finally, the ROM address is adjusted by a left shift operation.

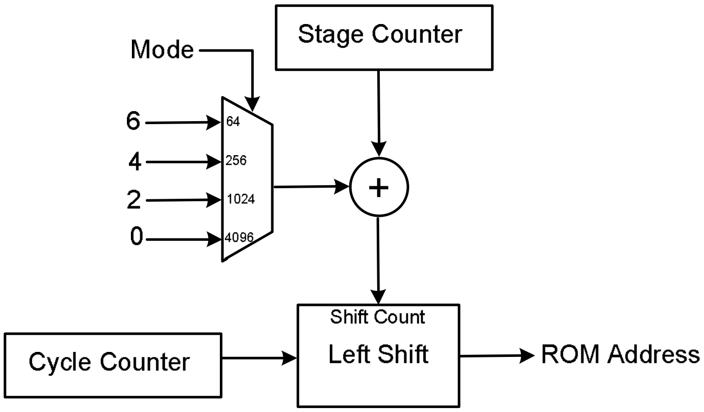


Figure 38. ROM Address Generation

The controller also generates the addresses of the two RAMs (see Figure 35): Switch_RAM_Address_0 (which connects to RAM0, RAM2, RAM4 and RAM6) and Switch_RAM_Address_1 (which connects to RAM1, RAM3, RAM5 and RAM7.) The RAM address generation is illustrated in Figure 40. The Switch_RAM_Address_0 is set to cycle_counter. The Switch_RAM_Address_1 is set to cycle_counter if the stage is a safe stage. Otherwise, the Switch_RAM_Address_1 is generated by inverting the upper (Stage_Counter-$\log_2 M$) bits of the cycle counter.

```
Rom_Address =0;
case ( mode  )
    `FFT_MODE_64__points: begin
        pe_id__cycle_q_2 =  { cycle_q[2:0], 5'b0 };
        Rom_Address  [2:0 ] =  pe_id__cycle_q_2  >> ( 5  - stage_q );
        Rom_Address  [8:0 ] = {Rom_Address  [2:0 ], 6'b00_0000};
    end
    `FFT_MODE_256_points: begin
        pe_id__cycle_q_2 =  { cycle_q[4:0], 7'b0 };
        Rom_Address  [4:0 ] =  pe_id__cycle_q_2  >> ( 7 - stage_q  );
        Rom_Address  [8:0 ] = {Rom_Address  [4:0 ], 4'b0000};
   end
    `FFT_MODE_1K__points: begin
        pe_id__cycle_q_2 =  { cycle_q[6:0], 9'b0 };
        Rom_Address  [6:0 ] =  pe_id__cycle_q_2  >> ( 9  - stage_q );
        Rom_Address  [8:0 ] = {Rom_Address  [6:0 ], 2'b00};
    end
    `FFT_MODE_4K__points: begin
        pe_id__cycle_q_2 =  { cycle_q[8:0], 11'b0 };
        Rom_Address  [8-1:0 ] =  pe_id__cycle_q_2  >> (11   - stage_q );
    End
endcase
```

Figure 39. Verilog Code for ROM Address Generation

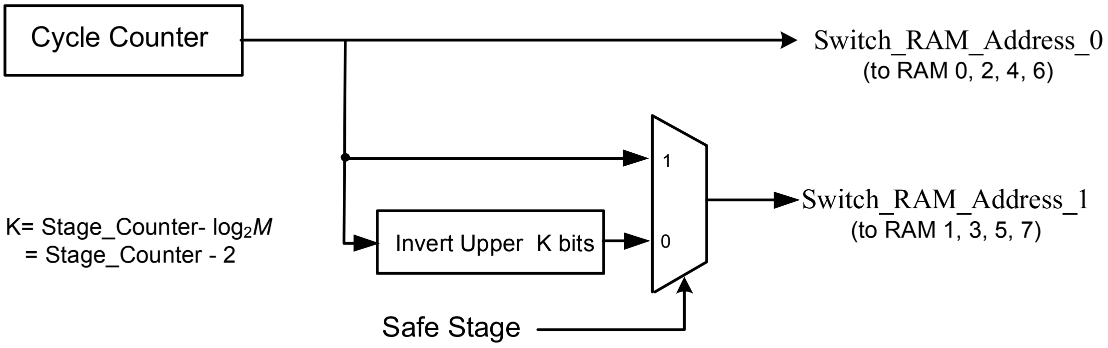


Figure 40. RAM Address Generation

The RAM address logic is illustrated by the Verilog code in Figure 41. Similar to the ROM address generation code, the RAM address generation logic configures itself based on the programmed mode (i.e., the value of *N*.) The generation of Address_0 is set to cycle_q (i.e., the cycle counter.) The Address_1 generation depends on the stage_q (i.e., stage counter). If the stage is safe-stage (i.e., stage_q less that or equal two), then Address_1 is equal to the cycle counter. If the stage is not safe-stage, Address_1 is set to a modified value of the cycle counter. The modified value is computed by inverting the upper (stage_q – 2) bits of cycle counter.

```
Addr0 = cycle_q;
if ( stage_q <=  `LOG_PE_NUM ) begin
    Addr1 = Addr0;
end
else begin
    Addr_invert_bits_tmp = { 9'h1ff, 9'h000};
    Addr_invert_bits_before_select = Addr_invert_bits_tmp [ stage_q + 6  :  stage_q - 2 ];
    case ( mode  )
        `FFT_MODE_64__points: begin
            Addr_invert_bits = Addr_invert_bits_before_select[8:6] >> 6 ;
         end
        `FFT_MODE_256_points: begin
            Addr_invert_bits = Addr_invert_bits_before_select[8:4] >> 4 ;
        end
       `FFT_MODE_1K__points: begin
            Addr_invert_bits = Addr_invert_bits_before_select[8:2] >> 2 ;
      end
      `FFT_MODE_4K__points: begin
            Addr_invert_bits = Addr_invert_bits_before_select[8:0] ;
      end
    endcase
    Addr1 = Addr0 ^ Addr_invert_bits;
end
```

Figure 41. Verilog Code for RAM Address Generation

At any given cycle, data flows between PE_{PE_ID} and two RAMs. The data MUXes (i.e., A, B and C in Figure 22) control the flow of data between the PE_{PE_ID} and the two RAMs. Figure 42 illustrates the select generation for A and B MUXes feeding PE_{PE_ID}. Figure 43 shows the select generation for the C MUXes feeding the RAMs. The W MUXes select logic is explained later.

```
case (stage_q)
  0: begin
      MUX_sel_for_PE_0 [pe_id] = { 1'b0, pe_id  };
      MUX_sel_for_PE_1 [pe_id] = { 1'b1, pe_id  };
    end
  1: begin
      MUX_sel_for_PE_0 [pe_id] = { pe_id[1], 1'b0, pe_id[0]  };
      MUX_sel_for_PE_1 [pe_id] = { pe_id[1], 1'b1, pe_id[0]  };
    end
    default: begin
      MUX_sel_for_PE_0 [pe_id] = { pe_id, 1'b0 };
      MUX_sel_for_PE_1 [pe_id] = { pe_id, 1'b1 };
    end
endcase
```

Figure 42. Verilog Code for PE MUX Select Generation

```
for (pe_id=0;  pe_id< `NUMBER_OF_PE; pe_id=pe_id+1) begin
   case ( MUX_sel_for_PE_0[pe_id])
     0 : MUX_sel_for_RAM_0 = pe_id;
     ...
     7 : MUX_sel_for_RAM_7 = pe_id;
   endcase
   case ( MUX_sel_for_PE_1[pe_id])
     0 : MUX_sel_for_RAM_0 = pe_id;
     ...
     7 : MUX_sel_for_RAM_7 = pe_id;
   endcase
end
```

Figure 43. Verilog Code for RAM Mux Select Generation

Processing Element (PE)

The PE serves as the ALU for the processor, where the radix-2 DIF butterfly operations are performed. An exploded view of the PE is shown in Figure 44. The input and output data are 64-bit double words, where the most significant 32-bits is the imaginary component and least significant 32-bits is the real component of the data. The 32-bit data format is two's complement fixed point, where the number of fractional bits is 16-bits.

The PE consists of six adders/subtractors (ADD), four multipliers (MUL), swap muxes and shuffle muxes. Figure 45 shows the swap and shuffle logic, which is part of the switch. The muxes perform the swap and shuffle operations. The adder and subtractor circuits are implemented using a carry lookahead adder. The multiplier is implemented using a Wallace multiplier. The Wallace multiplier is described in detail in the appendix. In case of an overflow, the results are saturated to either the maximum positive value $(7FFF\_FFFF)_{Hex}$ or the maximum negative value $(8000\_0000)_{Hex}$.

Significant numbers of the twiddle factors (*W*s) are either 1 (~25%) or -1 (15%). Hence, a potential power saving is to disable the multiplier whenever 1 or -1 is detected. An RTL power study was conducted to evaluate the following cases:

1. Normal design (no MUL disable),
2. Disabled MUL when W=1 or W=-1,
3. Disabled MUL when W=1.

The disabling of the multipliers was implemented by adding a set of MUXes at the input of the MUL. The MUXes allow the data to pass through when W=1 (or W=-1); otherwise, a value of zero is passed to the MUL. A summary of the average PE power is shown in Figure 46. The data is normalized to case (1). Unfortunately, the results indicate a power jump around 11-12% in case (2) and case (3), with case (3) being the worst. The question then is "How did a power saving idea end up hurting the overall power?" Further examination of the power reports reveals the reason for the power increase.

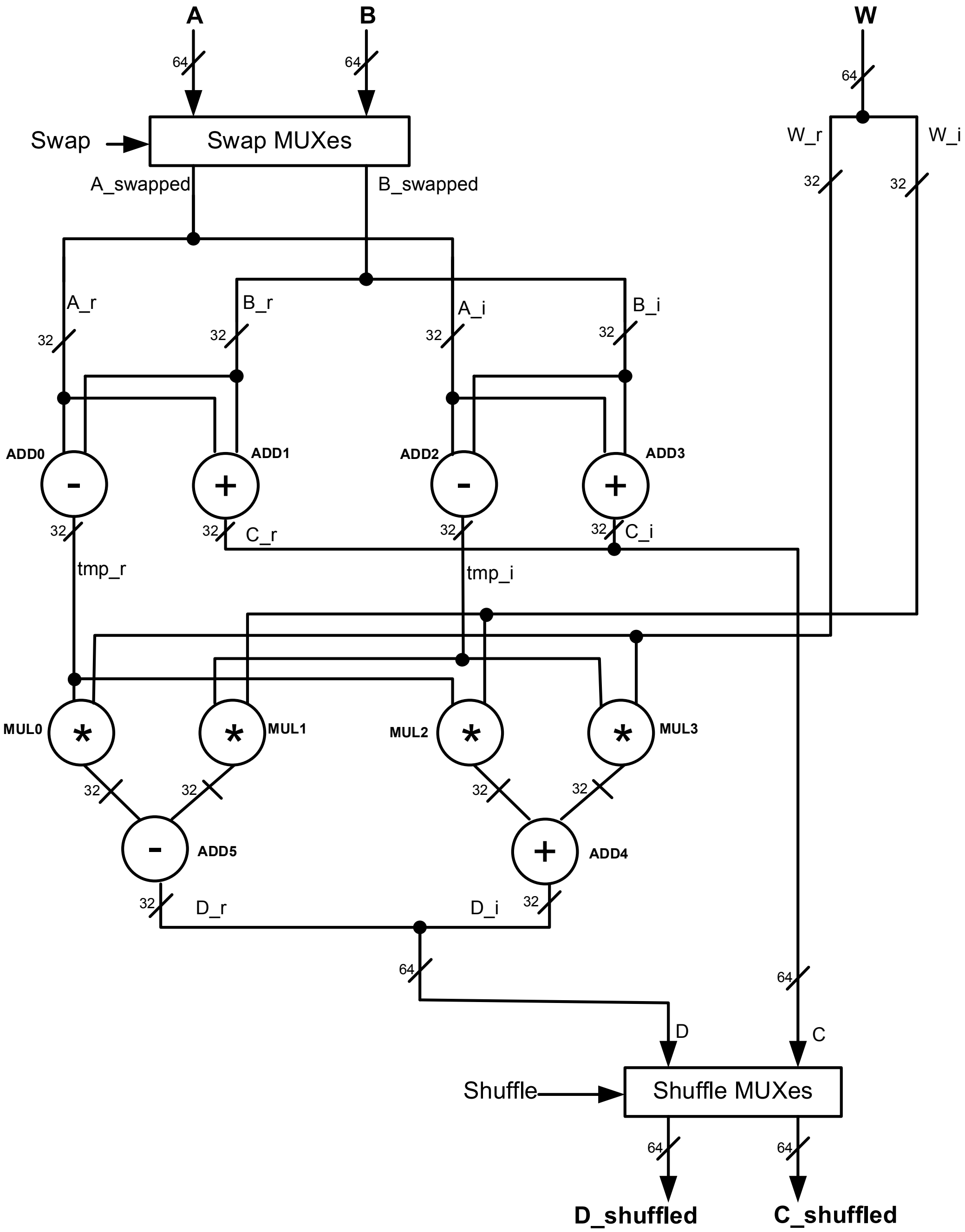


Figure 44. PE Block Diagram

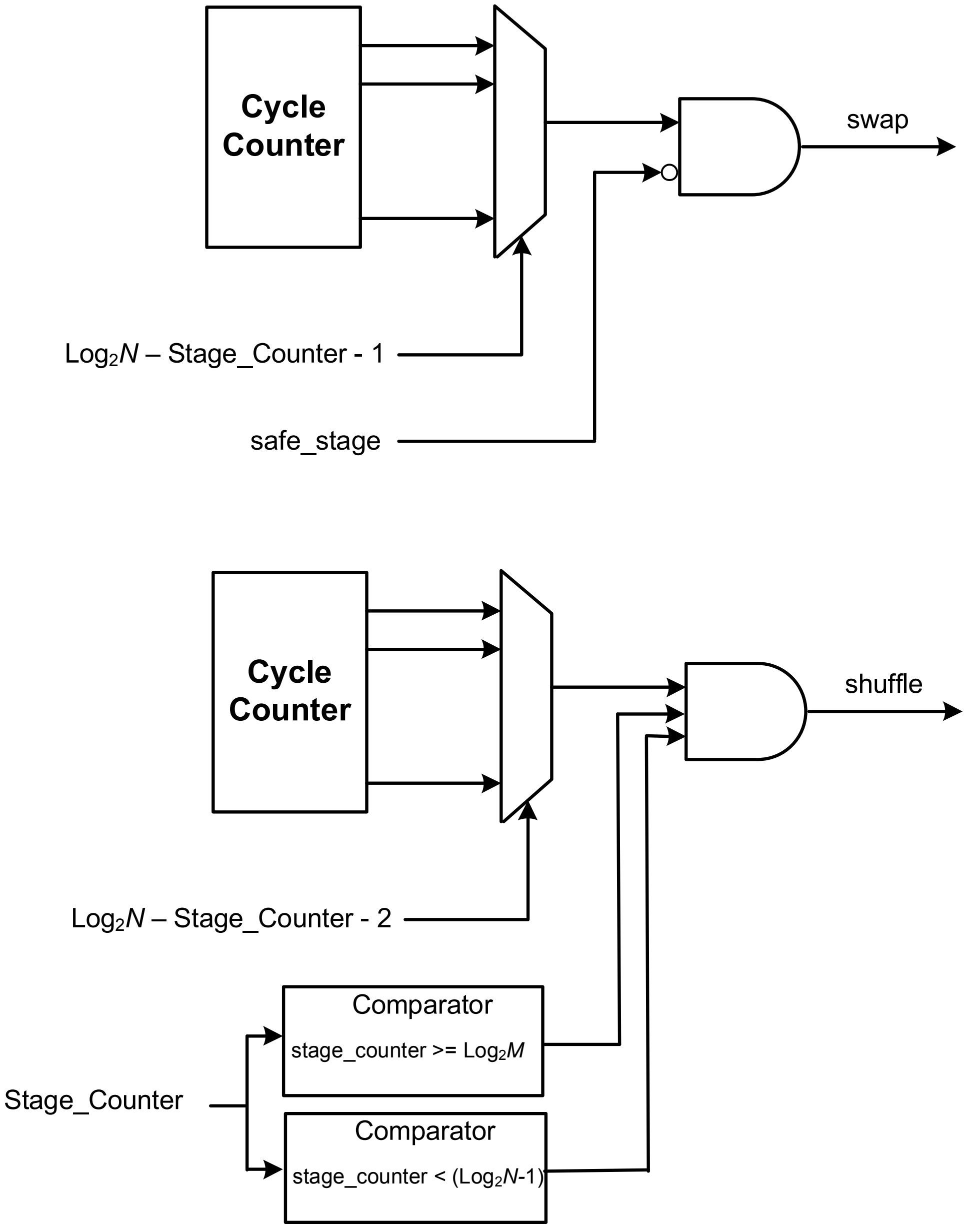


Figure 45. Shuffle and Swap Signal Generation

Figure 47 illustrates the PE power breakdown normalized to case (1). In case (1), 98% of the power is in the datapath (i.e., ADD/MUL) and 2% is in the random logic (e.g., MUXes.) In case (2) and (3), the datapath power dropped; however, the random logic power increased by more than the datapath power savings. The random logic power dissipation is mainly from the added MUXes. Those MUXes are in the critical path and have relatively large sizes. Moreover, they have a relatively high activity factor, and hence they dissipate high dynamic power. As a result, the idea of disabling the multiplier using MUXes was not implemented.

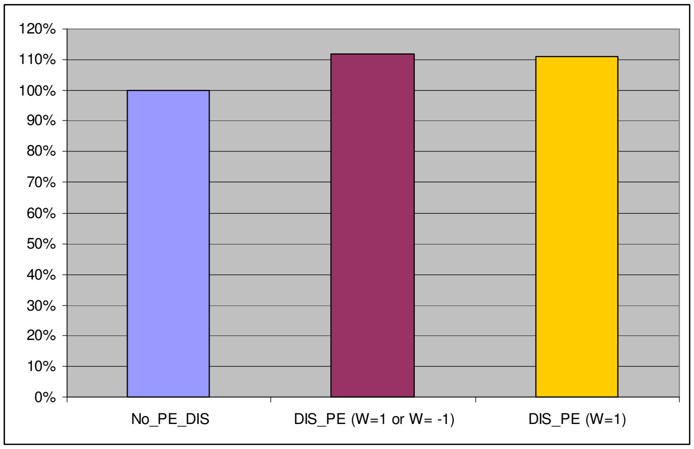


Figure 46. Normalized Power Results for Disabling the MUL

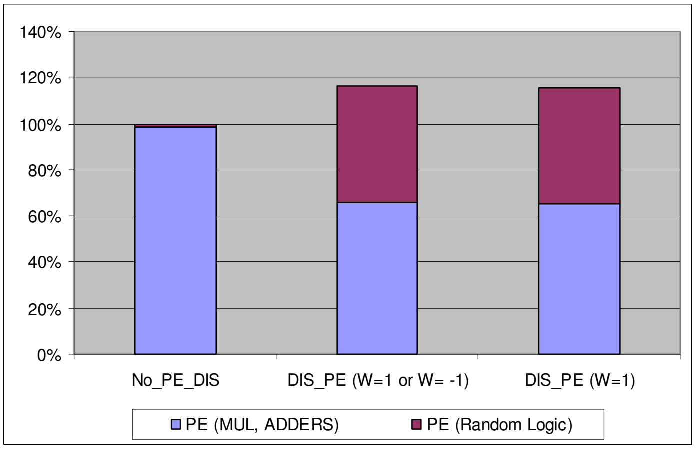


Figure 47. Breakdown of PE Power Across Experiments

ROM Design

The ROM units implement the twiddle factor table for W_{4096} points. The table contains 2048 entries; each entry contains 64-bit complex data. The upper 32-bits of the entry are the imaginary component and the lower 32-bits are the real component. Since the twiddle factor real and imaginary components have a range of [-1, 1], using 16-bit fractional format leaves the upper 15-bits unutilized. Another method is to use the 30-bit fractional format (i.e., 30-bits represent the fraction part of the number). In this format, the maximum positive twiddle value (i.e., W=+1) is represented by $(2000\_0000)_{Hex}$. Additionally, the maximum negative value is represented as $(C000\_0000)_{Hex}$. The format is two's complement as well. As a result of employing a 30-bit fractional format to represent the twiddle factor, the result of the PE multiplier is formed by selecting the product bits [61:30], as shown in Figure 48. The multiplier product has 64-bit, with the fixed point situated between bit 45 and bit 46.

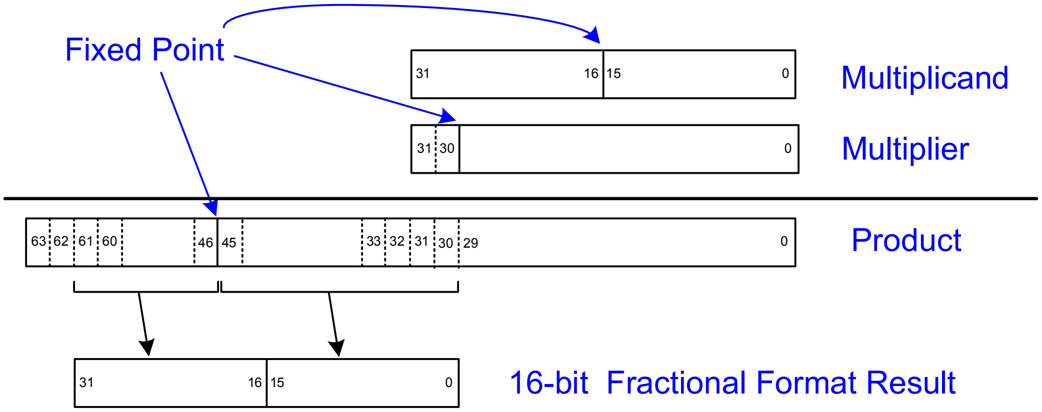


Figure 48. Generating 16-bit Fractional Format from the MUL output

Since each PE requires a twiddle input (i.e., ***W***), the design requires four ROM units. However, further investigation into ROM access reveals that it follows a methodical pattern that can be utilized to optimize the design. Assume that each ROM consists of four equal size quarters, where each quarter has 512 entries. It can be observed that:

- In each cycle, the four PEs access either one ROM or all ROMs,
- When all ROMs are accessed, the addresses are the same for all ROMs,
- When all ROMs are accessed, access addresses have same offset within a quarter,

These observations suggest the full sized ROMs can be replaced by smaller 512-entry ROMs. In fact, this rule can be generalized for any *M*-PEs design. The ROM size can be reduced such that:

- Each ROM has N/(2**M*) entries, for *N*-point FFT,
- ROM*i* stores the following twiddle factors: *i*N/(2*M)* to *(i+1)*N/(2*M)-1,*
- All ROMs share the same addresses.

The optimization not only shrinks the ROM size, but it also reduces the number of ROM address buses and lowers the dynamic power.

The data from ROM_j to PE_{pe_id} is routed through a set of MUXes as shown in Figure 35. The logic to create the MUX selects is illustrated in by the Verilog code in Figure 49. The logic configures itself based on the mode (i.e., *N*.) Initially, a temporary signal, pe_id__cycle_q_1, is created by concatenating the pe_id (i.e., PE ID), cycle_q (i.e., cycle counter) and two zeros. Next, the ROM_Sel is generated by right-shifting the temporary counter as demonstrated by the code. Finally, the ROM_Sel selects the right ROM data for PE_{pe_id}.

```
case ( mode  )
  `FFT_MODE_64__points: begin
     pe_id__cycle_q_1 = { pe_id[1:0] , cycle_q[2:0], 2'b00 };
     Rom_Sel [1:0]    =  pe_id__cycle_q_1  >> (5  -  stage_q );
  end
  `FFT_MODE_256_points: begin
     pe_id__cycle_q_1 = { pe_id[1:0] , cycle_q[4:0], 2'b00 };
     Rom_Sel [1:0]    =  pe_id__cycle_q_1  >> (7  -  stage_q);
  end
  `FFT_MODE_1K__points: begin
     pe_id__cycle_q_1 = { pe_id[1:0] , cycle_q[6:0], 2'b00 };
     Rom_Sel [1:0]    =  pe_id__cycle_q_1  >> (9  -  stage_q );
  end
  `FFT_MODE_4K__points: begin
     pe_id__cycle_q_1 = { pe_id[1:0] , cycle_q[8:0], 2'b00 };
     Rom_Sel [1:0]    =  pe_id__cycle_q_1  >> (11 - stage_q );
  end
endcase

case(Rom_Sel)
  0: Rom_RdData  = Rom_RdData_0;
  1: Rom_RdData  = Rom_RdData_1;
  2: Rom_RdData  = Rom_RdData_2;
  3: Rom_RdData  = Rom_RdData_3;
endcase
```

Figure 49. Routing ROM Data to PE$_{pe\_id}$

Various circuit ROM implementations are discussed in [30] and [31]. One class of the implementations employs a grounded-gate PMOS transistor to charge the ROM array. This implementation suffers from static leakage through the PMOS transistor. Another class utilizes a clocked-PMOS transistor to precharge the array. However, the clock load and power is higher in this design, which is not desirable for low power implementation. Another option is to implement the ROM using static CMOS. This solution is attractive because it does not increase the clock power or static leakage. Moreover, because the size of the ROM is relatively small and the timing of accessing the ROMs is almost a cycle and a half, the gate sizes are relatively small. Figure 50 illustrates the ROM access timing path. The path starts at the rising edge of the clock of the RAM-Read micro-cycle by generating the ROM address in the switch. Next, the address is decoded in the ROM. The ROM read data is then generated after the rise of the clock of the PE micro-cycle. The data is delivered to the MUL just before the subtractor output arrives at the MUL, see Figure 44. In addition, the ROM data is valid until the rise of the Write-RAM micro-cycle clock. One issue that arises from static implementation is that whenever the address changes, the ROM logic toggles, this results in an increase in the ROM dynamic power. To mitigate this issue, the address lines are kept quiet and toggled only when the ROMs are accessed. This is achieved by data-gating the ROM address lines in the switch logic. As a result, a significant drop in power (57%) was accomplished as demonstrated by Figure 51. The figure illustrates the ROM power with and without data gating.

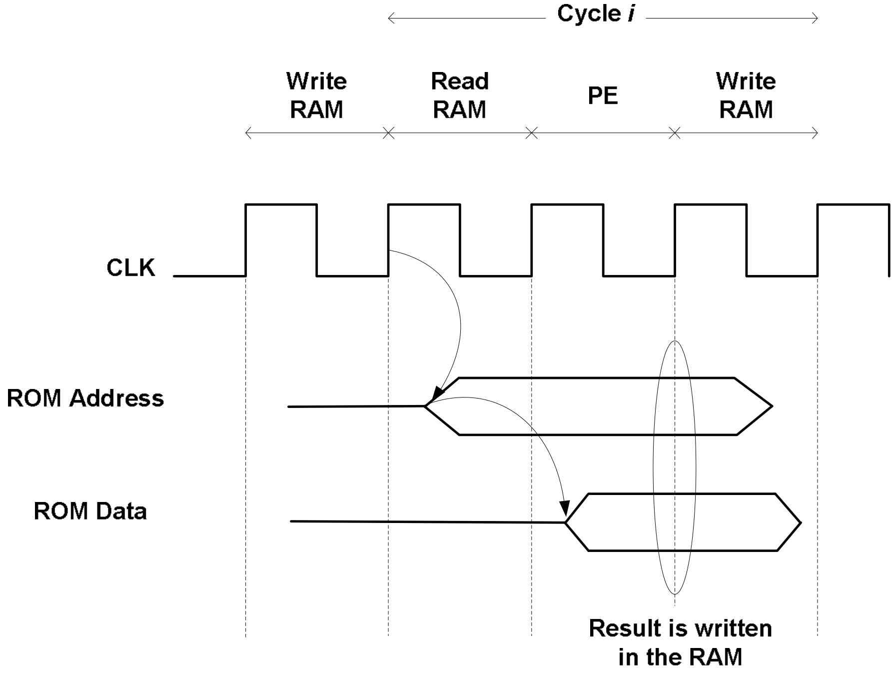


Figure 50. ROM Access Timing Path

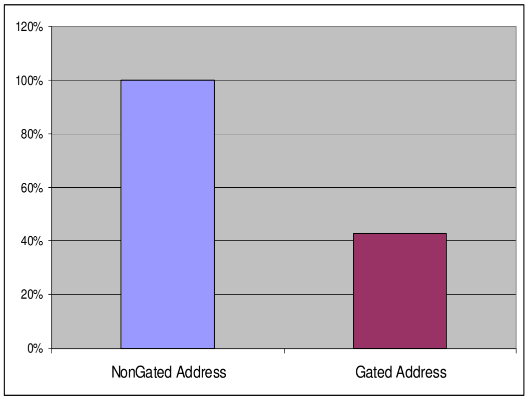


Figure 51. Reducing ROM Power using Address-Gating

Register Unit

The register unit contains two types of registers: control and status. The control registers are used by the state machines and the address generation hardware. The status register stores the machine status which is checked by various parts of the processor. Table 40 describes the registers.

Table 40. Register Specifications

Register	Type	Field	Description
Register 0	Control	[1:0]	Number of active PEs
		[6:2]	Number of blocks
		[9:8]	Number of points (64, 256, 1024 or 4096)
Register 1	Status	[0]	Completed FFT point
		[1]	Completed FFT block
		[2]	Busy
		[3]	Set_0
Register 2	Control Register	[11:0]	Upper 12-bits of the source block
Register 3	Control Register	[11:0]	Upper 12-bits of the destination block

5.4 Configuration and Operation

The processor communicates with the outside system via three main operations: configuration, read and write. The configuration operation programs the control registers, listed in Table 40 . The programming includes setting the number of active PEs. Also, the operation sets up the number of blocks to be processed (up to 31 blocks) and the number of FFT points (i.e., *N*). Moreover, the configuration operation programs the source and destination addresses of the processed block (i.e., register-2 and register-3).

The timing diagram for the configuration is illustrated in Figure 52. When fft_config is asserted by the external system, the configuration vector is latched on the same rising clock. The processor requires a minimum of two cycles to configure itself. The external system can then assert fft_start, which signals the processor to begin reading the first block. The configuration vector is 64-bits; however, not all the bits are utilized. Table 41 illustrates the interpretation of the configuration vector.

Table 41. Configuration Vector

Control Vector Bits	Parameter	Description
[1:0]	Active PEs	0: one PE is active 1: two PEs are active 2: three PEs are active 3: four PEs are active
[6:2]	Number of blocks	
[9:8]	Number of points	0: 64 points 1: 256 points 2: 1024 points 3: 4096 points
[21:10]	Source offset address	Upper 12 bits of source address
[33:22]	Destination offset address	Upper 12 bits of destination address

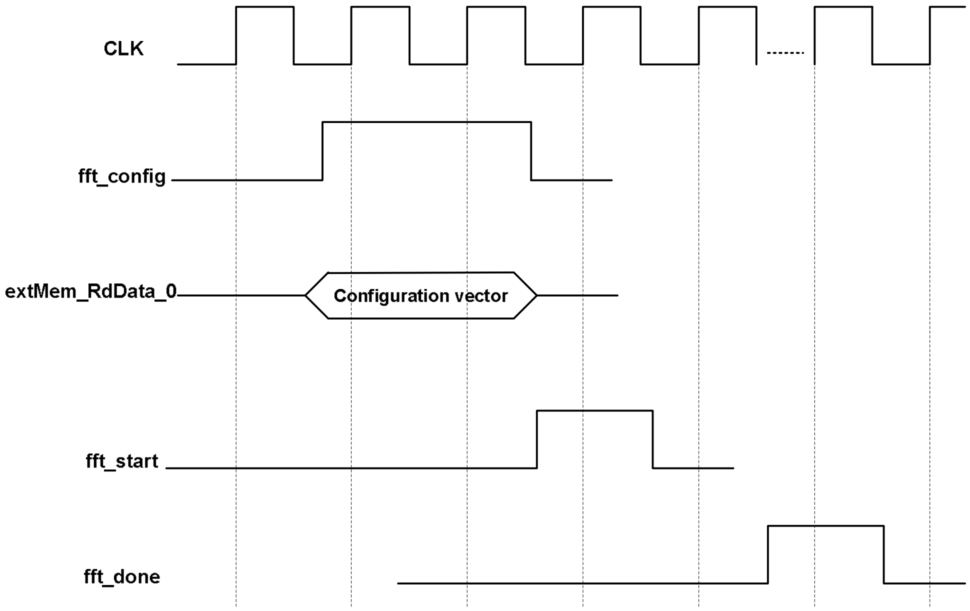


Figure 52. Processor Configuration Timing Diagram

The read operation loads data from the external memory to the designated RAM set. The write operation writes the data from the RAMs to the external memory. The design assumes the following features about the external memory:

- Operates synchronously with CLK,
- Capable of back to back reads and back to back writes.

Since processing the block takes much longer than reading/writing the block from/to external memory, the read/write latency can be more than one cycle. In fact, the latency can be as high as four cycles before impacting the processor performance. This design assumes a latency of one cycle.

The timing diagrams for read and write operations are shown in Figure 53 and Figure 54. The read operation starts by asserting extMem_SEL=1, extMem_WR=0 before the rise of the clock. The processor latches the read data on the next rising edge of the clock. Furthermore, the Write operation starts by asserting extMem_SEL=1, extMem_WR=1. Concurrently, write data is asserted in the extMem_WrData buses.

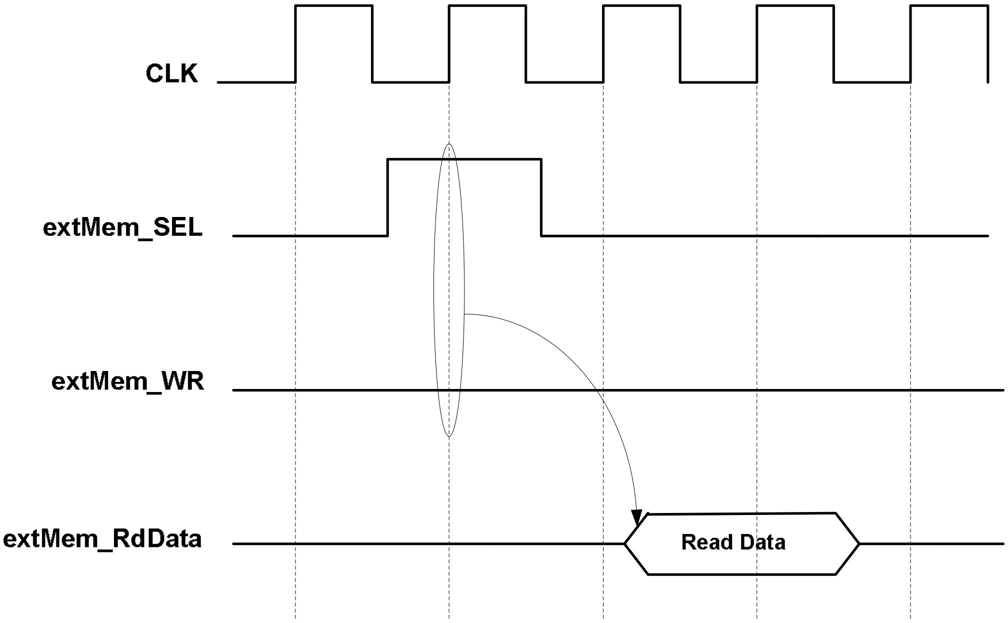


Figure 53. External Memory Read Access

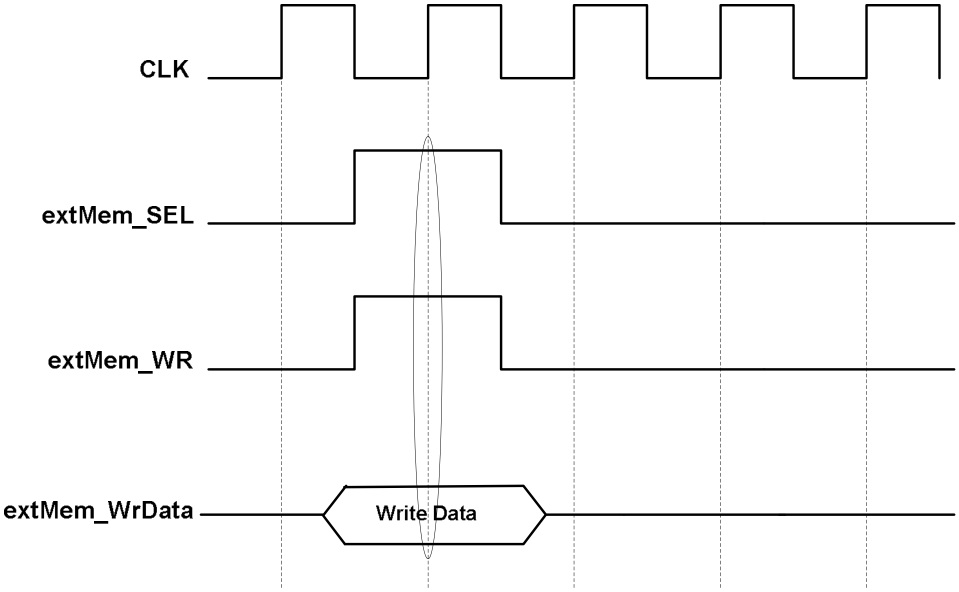


Figure 54. External Memory Write Access

5.5 Hardware and Timing Complexities

The hardware complexity of the design is summarized in Table 42. The first column lists the main design components. The second column estimates the complexity of the component. The third column indicates the number of the components in the design. The fourth column presents the total component complexity.

Table 42. Hardware Complexity

Component	**Complexity of One Component**	**Number**	**Total complexity**
PEs	4 32-bit×32-bit Multipliers 6 32-bit Adders	4	16 32-bit×32-bit Multipliers 24 32-bit Adders
Switch	5*xM* 64-bit MUX [$\text{Log}_2(\text{Log}_2 N)$]-bits Stage Counter [$\text{Log}_2 N$- Log_2M -1]-bits Cycle Counter	1	9*xM* 64-bit MUX [$\text{Log}_2(\text{Log}_2 N)$]-bits Stage Counter [$\text{Log}_2 N$- Log_2M -1]-bits Cycle Counter
ROM	N/8 × 64-bit	4	*N/2* × 64-bit
RAMs	N/8 × 64-bit	8	*N* ×64-bit
ExtMem Controller	20-bit Adder $\text{Log}_2 N$-1 bit counter	1	20-bit Adder $\text{Log}_2 N$-1 bit counter

The Timing delay for an N-point computation depends on the time to load blocks, compute the FFT and write back the results. Without impacting the final analysis, the time to load the initial block and time to write last block are ignored, since they are short and occur once during a block execution. Recall that there are $\log_2 N$ stages and each stage has N/8 cycles, as illustrated in Figure 37. When all PEs are active, the cycle has three clocks, during which the PEs executes twelve operations (i.e., Read RAM, PE and Write RAM). If only one PE is active, the cycle has twelve clocks. Therefore, the cycle contains 12/(Number_of_Active_PEs) clocks. Hence, the number of clocks to process an N-point FFT is:

$$T = \left[\frac{12}{Number\ of\ \ Active\ PEs} \times \frac{N}{8} \right] \log_2 N$$

Table 43 computes the number of clocks and the samples/second for different FFT configurations when all four PEs are active.

Table 43. Number of Clocks for Different FFT Configurations

N	Stages	Cycles per Stage	Clocks per Cycle	Total Clocks	Samples/Second
64	6	8	3	144	89 MSPS
256	8	32	3	768	67 MSPS
1024	10	128	3	3840	53 MSPS
4096	12	512	3	18432	44 MSPS

5.6 Summary

This chapter presented the logic design of the non-pipeline switch-based FFT processor. The design specifications were presented. Then, the functional blocks and operations were examined. Lastly, the design complexity was analyzed.

Chapter 6

RTL Simulation and Synthesis

After the RTL specifications were completed, the design was coded in Verilog. To validate its behavior, functional simulations were performed on the design as illustrated in Figure 19. Once verified, the RTL code was synthesized into gate-level netlist. The timing and power behaviors of the gate-netlist were analyzed. This chapter summarizes the front-end design simulations and analysis. The first section discusses the simulation results performed on the RTL design. The results are presented through a series of frequency spectrum plots. The second section examines the static timing analysis of the synthesized RTL. Lastly, a power analysis for the gate-level design is presented.

6.1 RTL Simulation Results

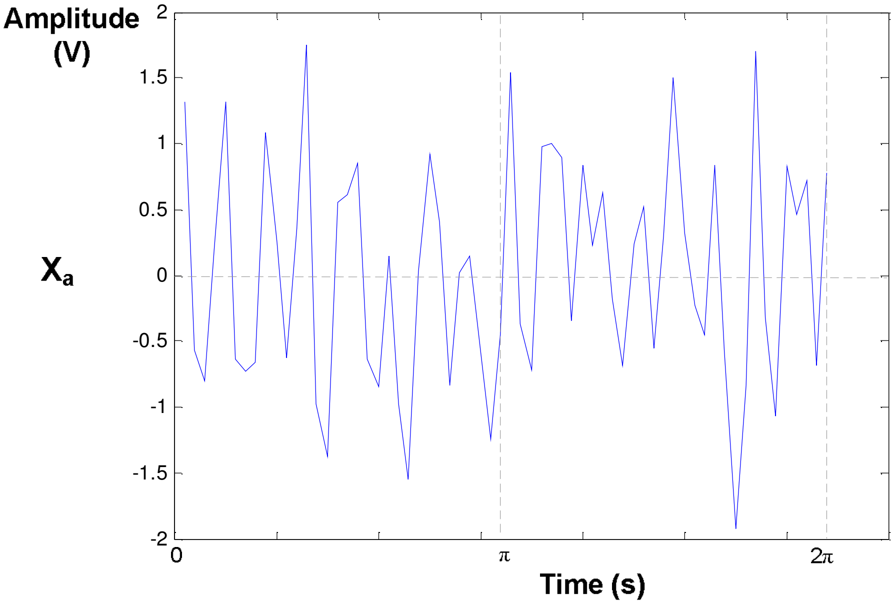
The results of the FFT processor are best illustrated in frequency-domain plots. The following discussion is based on two time-domain test signals. The first signal (x_a) has one frequency component of 16 Hz. The second signal (x_b) has two frequency components: 16 Hz and 64/14 Hz. A noise signal is added to x_a and x_b. The noise signal is a uniformly distributed random numbers between +1 and -1:

$$x_a(t) = \cos(2\pi * 16t) + \text{random-noise}$$

$$x_b(i) = \cos(2\pi * 16t) + \cos(2\pi * (64/14)*t) + \text{random-noise}$$

The time-domain plots of the signals are shown in Figure 55 . To reduce spectral leakage, the input signals are processed utilizing a Hamming window before FFT processing [43].

Next, the processor computed the FFT values for the test signals. The results are then plotted in the frequency domain as shown in Figure 56-Figure 59. The plots are for 64-, 256-, 1024- and 4096-point FFTs. Part (a) of the plot is for x_a and (b) is for x_b.



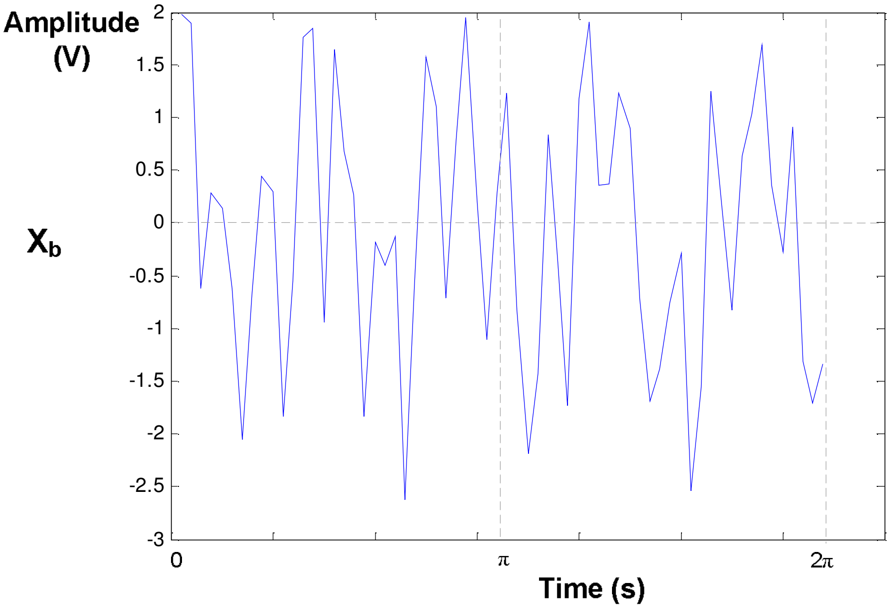


Figure 55. Test signal x_a and x_b

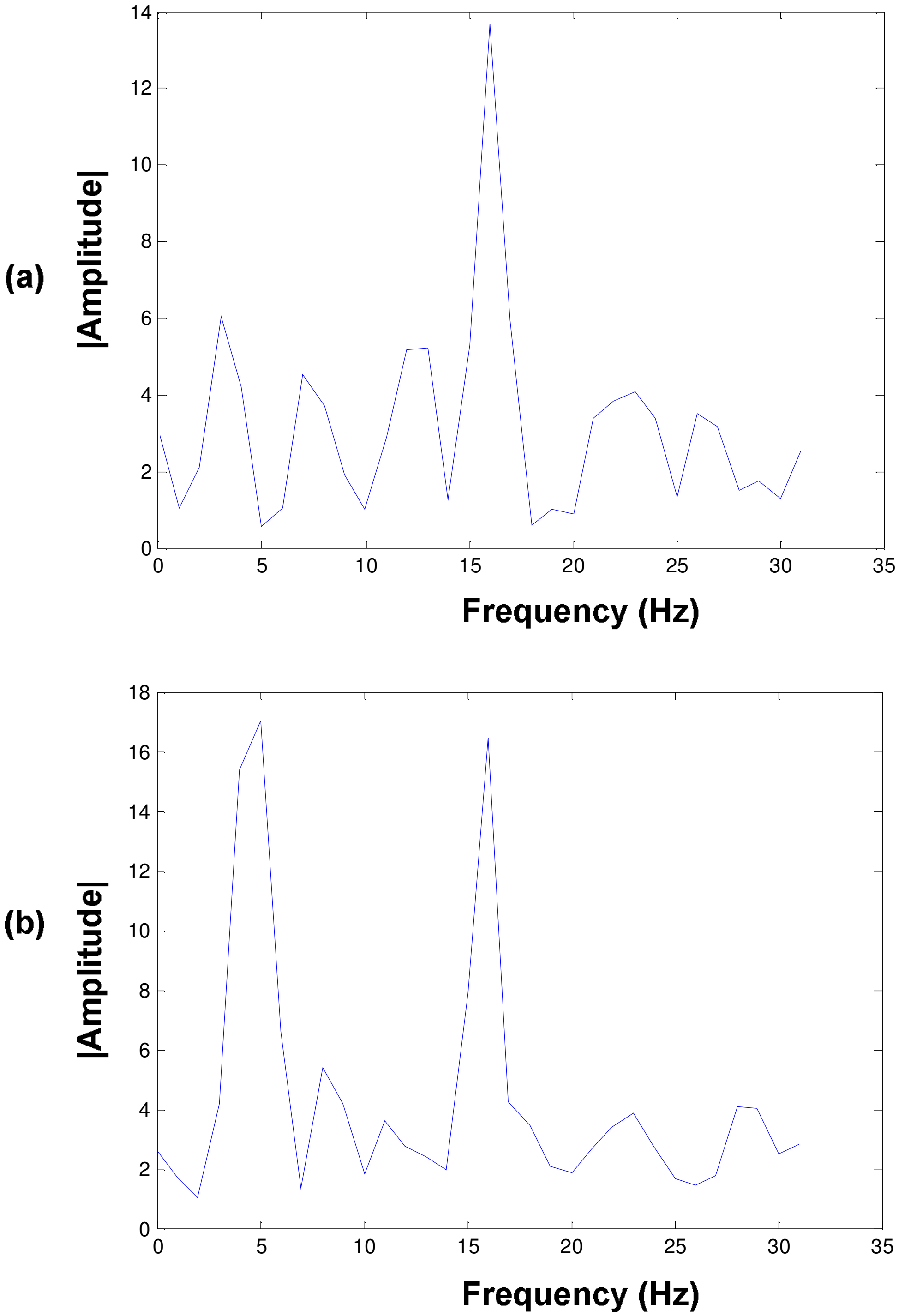


Figure 56. 64-point RTL-Generated FFT

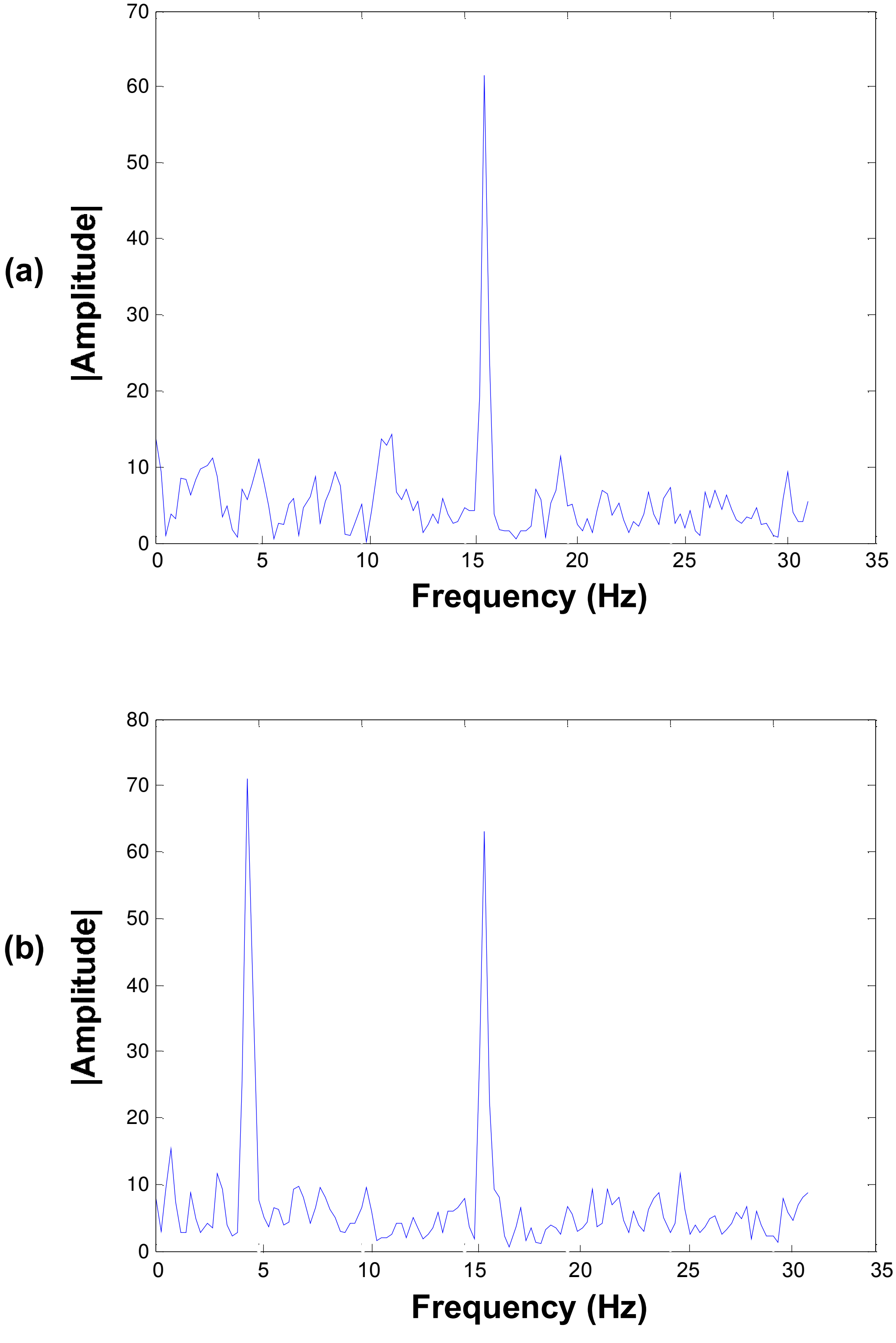


Figure 57. 256-point RTL-Generated FFT

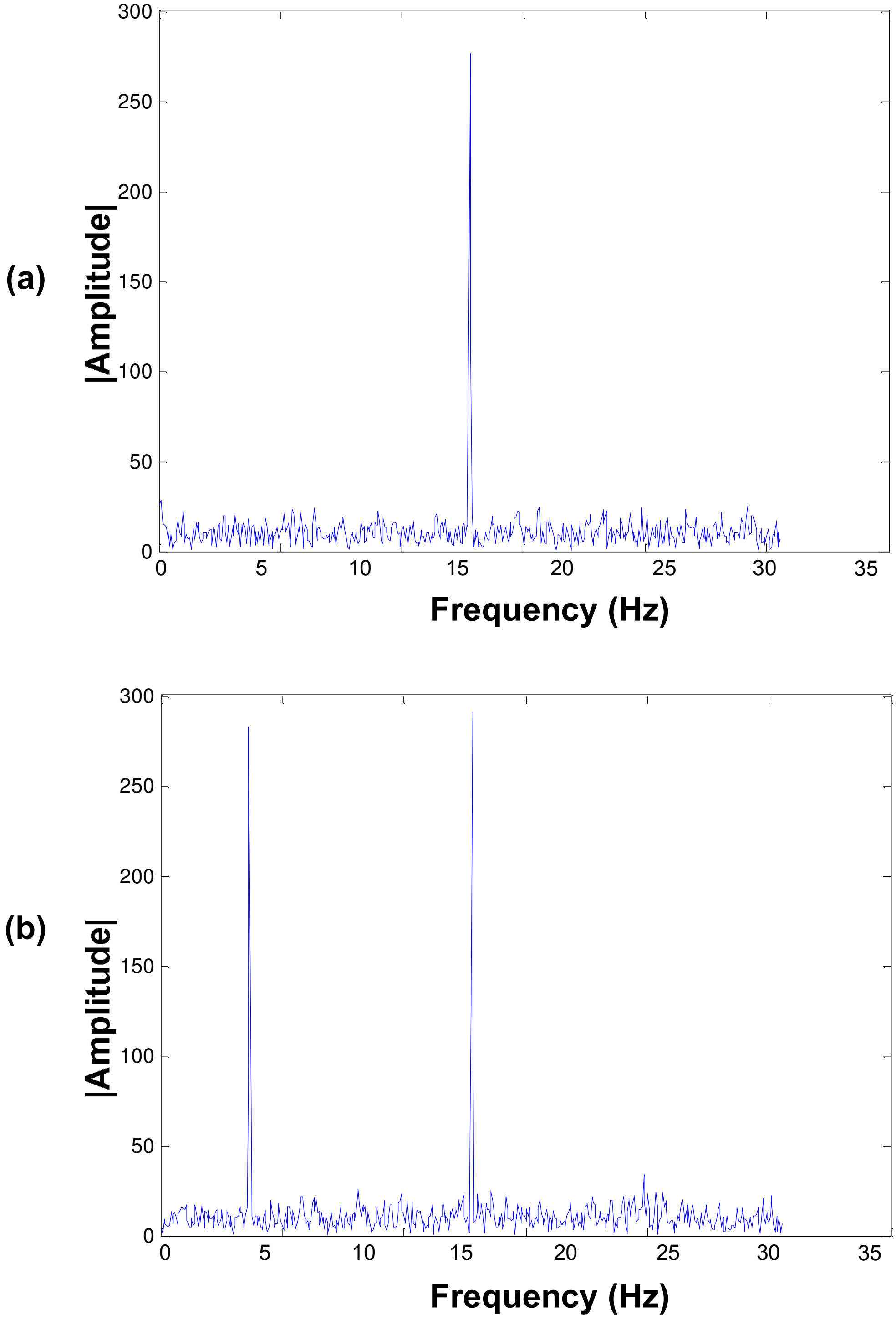


Figure 58. 1024-point RTL-Generated FFT

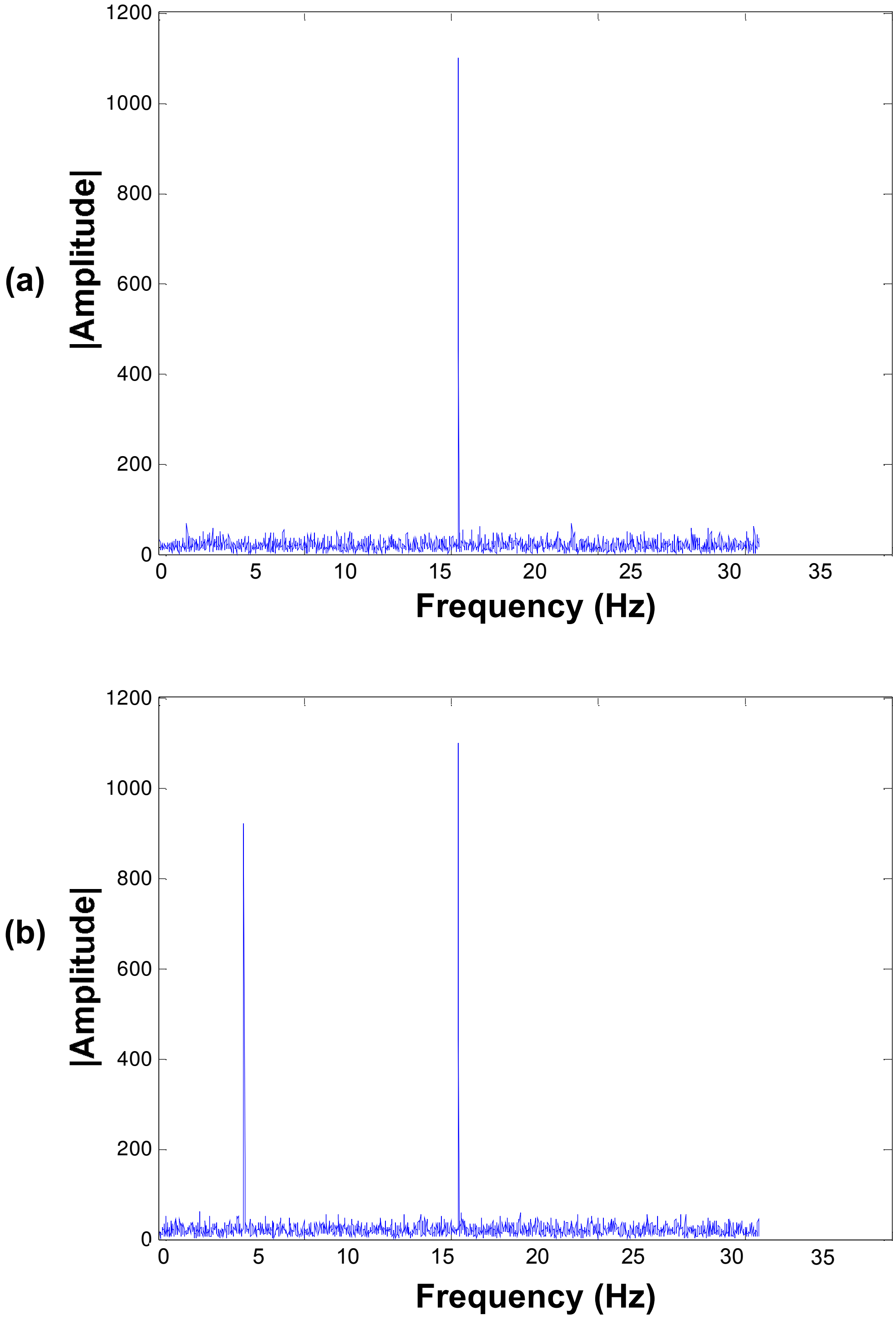


Figure 59. 4096-point RTL-Generated FFT

The amplitude of the FFT bins (containing the signal tones) depends on the number of FFT points (i.e., *N*). Hence, a larger *N* has a higher processing gain. The Signal-to-Noise-Ratio (SNR) can be described as the output signal-power over the average noise-power level [28]. The output noise is proportional to square-root of *N*. However, the output magnitude of the bin (containing the signal tone) is proportional to *N*. Therefore, if $N_1>N_2$, the SNR_{N1} is expressed as ([28]) :

$$SNR_{N1} = SNR_{N2} + 20\log_{10}\left(\sqrt{\frac{N_1}{N_2}}\right)$$

For example, if N_1 is four times N_2, then the SNR_{N1} improves by 6.02 dB.

The above conclusion can be also derived from the plots. Table 44 summarizes the difference between the FFT bins (containing the signal tone) and the noise floor. The difference is expressed in absolute magnitude (second column) and dB (third column.) The results indicate that by increasing *N* by a factor of 4X, the noise floor is pushed down (and therefore improving SNR) by an order of 6 dBs.

Table 44. Noise Floor for Various FFT Plots

N	**$\lvert A_m$ – Noise_Floor$\rvert$**	**$\lvert A_m$ – Noise_Floor$\rvert_{(dB)}$**
64-point	15	12 dB
256-point	60	18 dB
1024-point	250	24 dB
4096-point	1000	30 dB

Another observation is that the output bin representing the tone at 64/14 Hz has leaked into the neighboring bins. The frequencies that are represented in the bins must satisfy:

$$F(m) = \frac{mf_s}{N}, \; where \; m = 0,1,...,N-1$$

In Figure 56 (b), f_s=64 Hz, and therefore frequency 64/14=4.5714 Hz is not represented by any bin. In fact it has leaked to all the bins; however, it has leaked the most into bin 4 and bin 5.

6.2 RTL Synthesis

The RTL code was synthesized with exception of the RAMs. As discussed in the last chapter, RAMs were modeled using a library model. The synthesis was constrained to a five nanosecond (ns) clock period with the minimum area possible. The rest of the section presents the timing and power analysis performed on the synthesis-generated gate-level netlist.

6.2.1. Timing Analysis

The timing analysis was based on static timing analysis (STA), which computes the timing paths statically (i.e., without vector-based dynamic simulations.) The analysis computes the largest (and minimum) delays between flip-flops, input-pins to the inputs of flip-flops and flop-flop outputs to the output pins. The timing constraints are provided to the analysis, which includes the clock period, input pin arrival times and the output pin required times. Below is a detailed discussion of three of the critical paths in the design.

The worst critical path in the design is the timing path thorough the switch sub-system (shown in Figure 60), which occurs in the PE micro-cycle (i.e., clock.) During this clock several circuit activities must be completed, including:

- The RAM clock-to-Q delay, which is equal to the flip-flop clock-to-Q delay since the read data is driven by the flip-flop output,
- Switch muxes when routing the read data to the destination PE ,
- 32-bit subtractor,
- 32-bit multiplier,
- 32-bit adder,
- Switch muxes to route results from PE to destination RAM,
- The RAM setup time.

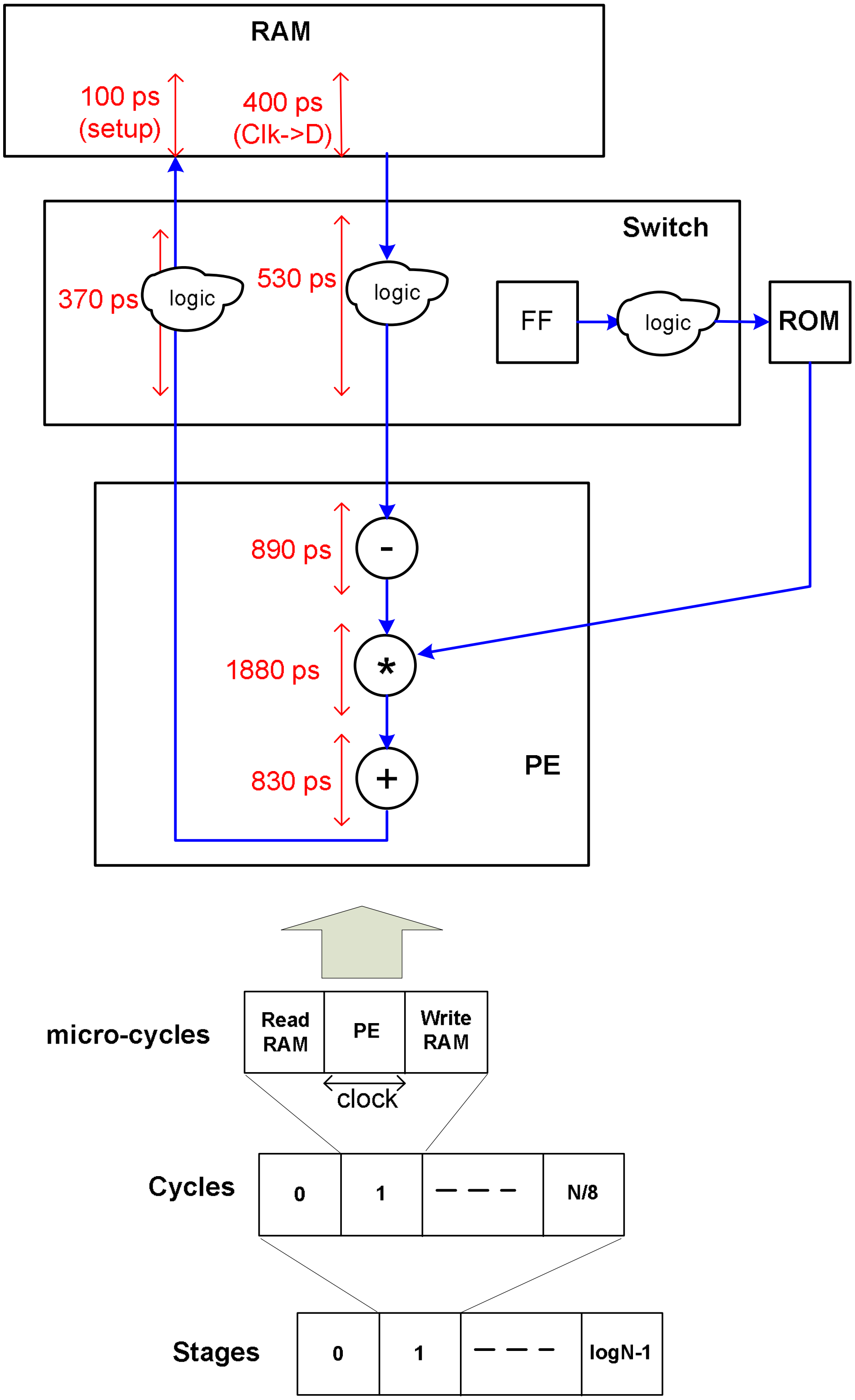


Figure 60. PE Timing Path

Table 45 presents the timing delays for the above operations. The first column lists the operation. The second and third columns show the operation delay in absolute value (in ps) and as a percentage of the total path delay. The fourth column indicates the number of logic levels in the operation. The last column computes the delay (in ps) per logic level. Clearly, the design meets the clock period goal, which is equal to five ns (i.e., the design frequency is 200 MHz.) Further examination of the table reveals the following observations. First, the delay per gate is 100 ps, which is a relatively high delay for gates realized with the 65nm-process. The reason for the high delay is because the target process is a low-power (specifically low-leakage) process which is much slower than a high performance processes. If a high performance process was used, a target of one ns for the clock period is likely to be achievable.

Secondly, the delay overhead of the switch is 18% of the total delay, which is less than one-fifth of the clock period. This indicates that the switch component does not dominate the delay. Furthermore, the design has an opportunity to expand the number of PEs with only a minor increase in the delay, since any increase in the switch delay will not severely impact the critical path.

Table 45. PE Timing Path

Operation	Delay	% of Total Delay	Logic Levels	Delay/Gate
Drive RAM Data	400 ps	8%	3	133.3 ps
Switch Delay 1	530 ps	11%	5	106 ps
32-bit Subtractor	890 ps	18%	12	74.2 ps
32-bit Multiplier	1880 ps	38%	16	117.5 ps
32-bit Adder	830 ps	17%	9	92.2 ps
Switch Delay 2	370 ps	7%	5	74 ps
RAM setup	100 ps	2%		
Total Path	5000 ps	100%	50	100 ps

The ROM access path is next to be analyzed. This path is a two-cycle path that spans the Read-RAM and PE micro-cycles (i.e., clocks.) Table 46 presents the main operations and their delays in ROM path. The table columns are organized similarly to the last table. Examining the table indicates that:

- The timing path has a positive slack of 20 ps. The ROM output arrives at the multiplier inputs just a little earlier than the subtractor output.
- Because the path has positive slack, the synthesis did not have to upsize the gates in the ROM path i.e., many gates are implemented using small gates. This is clear from the fact that the average delay/gate is higher for this path than the PE path.
- The number of switch logic levels to route the ROM data into PEs (four logic levels) is less than the number of switch logic levels to route the RAMs data into the PE (five levels.) This is because the number of RAMs is double the number of ROMs.

Table 46. ROM Access Timing Path

Operation	**Delay**	**% of Total Delay**	**# Logic Levels**	**Delay/Gate**
Address Generation	2270 ps	23%	15	151.3 ps
ROM Delay	3930 ps	79%	17	231.2 ps
Switch Delay 1	730 ps	15%	4	182.5 ps
32-bit Multiplier	1560 ps	31%	16	97.5 ps
32-bit Adder	1030 ps	21%	13	79.2 ps
Switch Delay 2	360 ps	7%	5	72 ps
RAM setup	100 ps	2%		
Total Path	9980 ps	200%	70	142.6 ps

Finally, the memory sub-system timing paths are not as critical as the switch sub-system, mainly because there are no large datapaths. As mentioned in the previous chapter, the controller state machine starts by reading the first block, followed by states which read the rest of the blocks, see Figure 32. Each state consists of N/8 micro-cycles (i.e., clocks.) During each clock, the external memory address is generated thorough the logic illustrated in Figure 61. Initially, the “current state” flip-flops is evaluated to generate the “increment-counter” signal. Next, the counter is updated and its output is carefully selected through a 4-1 MUX based on the mode (i.e., *N*.) The output of the MUX is then added to the offset using a 20-bit adder.

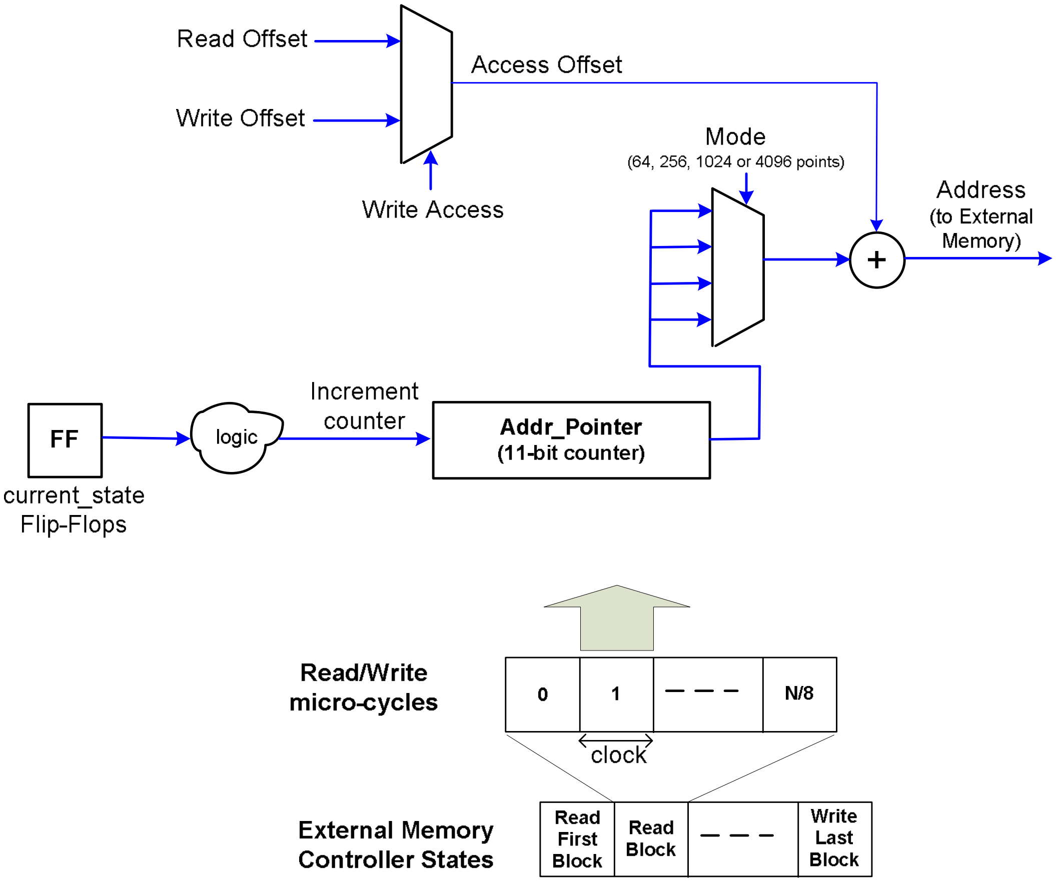


Figure 61. External Memory Controller Timing Path

The timing delays of the path are presented in Table 47. The table columns are organized similarly to previous path delay tables. The timing path requires half a clock period, which is the timing constraint provided to the synthesis tool. Observe that the adder is implemented as ripple carry adder, since it is small and there is enough timing margin for its delay. Moreover, the delay/gate is smaller than other paths because the logic has smaller fanout (i.e., loads) than the previous paths. This results in a faster delay through the small-sized gates.

Table 47. External Memory Address Generation Timing Path

Operation	Delay	% of Total Delay	# Logic Levels	Delay/Gate
FF clk->Q Delay	290 ps	12%	3	96.7 ps
11-bit counter	800 ps	32%	7	114.3 ps
20-bit adder	1410 ps	56%	20	70.5 ps
Total Path	2500 ps	100%	30	83.3 ps

6.3 Power Estimation

Power simulations performed on a synthesized RTL provides quick feedback about the design power. Postponing the power estimation until after layout increases the feedback delay, making it harder to enhance the RTL power. Furthermore, tracking RTL power regularly helps build a power trend for the design. Another advantage for RTL power estimates is that it enables experimentations with different micro-architectures, e.g., is forwarding the datapath output more power efficient? Finally, RTL power simulation results are easier to understand than the layout power estimates. Often, the physical design tool flattens the hierarchy, changes net names and modifies the gates. And as a result, it is harder to associate the layout power reports to the original design.

Power Simulations:

The synthesized gate-level netlist was simulated using the test signals shown in Figure 55 –part (a) with N=1024, and with no windowing applied (for easier comparison with other designs.) Four power simulations were conducted, each with different number of active PEs. Each cell power is reported as shown in Figure 62. The internal power represents the power dissipation in the internal cell nodes. It also includes the leakage power. The switching power is the power dissipated due to switching the cell capacitive load.

The results of the simulations are presented in Table 48. The major columns represent the cases of: one-PE, two-PE, three-PE and four-PE. Within each major column, there are three minor columns representing internal power, switching power and total power. The rows represent the major design components: Switch, External Memory Controller, Registers, ROMs, RAMs (set-0) and PEs. Since its power is high, the power for the PE is broken further into the major datapaths. The names of the PE components correspond to the names used in Figure 44. The data from the table is used in the below power analysis.

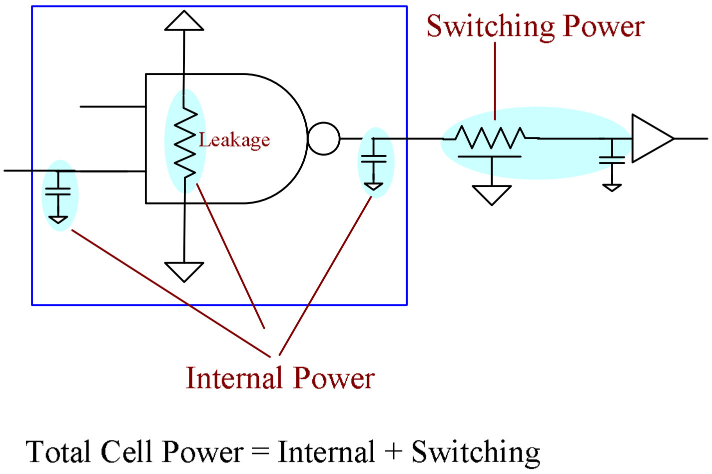


Figure 62. Reporting Cell Power

Table 48. Post Synthesis Power Results

Component	1-PE			2-PE			3-PE			4-PE		
	I	S	T	I	S	T	I	S	T	I	S	T
switch	0.50	0.50	0.99	0.79	1.04	1.83	1.12	1.60	2.72	1.40	2.10	3.49
extMemCtl	0.05	0.00	0.05	0.05	0.00	0.05	0.05	0.00	0.05	0.05	0.00	0.05
registers	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.00	0.01
rom0	0.03	0.09	0.12	0.05	0.16	0.20	0.04	0.15	0.19	0.05	0.19	0.24
rom1	0.03	0.08	0.11	0.05	0.16	0.20	0.05	0.16	0.20	0.06	0.21	0.26
rom2	0.03	0.09	0.12	0.05	0.16	0.21	0.04	0.15	0.19	0.05	0.19	0.25
rom3	0.03	0.09	0.12	0.05	0.16	0.21	0.05	0.16	0.21	0.06	0.22	0.28
ram_0_0	0.88	0.01	0.88	1.75	0.02	1.77	2.66	0.03	2.69	3.59	0.04	3.63
ram_0_1	0.88	0.01	0.88	1.75	0.02	1.77	2.66	0.03	2.69	3.59	0.04	3.63
ram_0_2	0.88	0.01	0.89	1.75	0.02	1.77	2.66	0.04	2.69	3.59	0.05	3.63
ram_0_3	0.88	0.01	0.89	1.75	0.02	1.77	2.66	0.04	2.69	3.59	0.05	3.63
ram_0_4	0.88	0.01	0.89	1.75	0.02	1.78	2.66	0.04	2.69	3.59	0.05	3.64
ram_0_5	0.88	0.01	0.89	1.75	0.02	1.78	2.66	0.04	2.69	3.59	0.05	3.64
ram_0_6	0.88	0.01	0.89	1.75	0.02	1.78	2.66	0.04	2.69	3.59	0.05	3.64
ram_0_7	0.88	0.01	0.89	1.75	0.02	1.78	2.66	0.04	2.69	3.59	0.05	3.64
pe_0/mul0	1.17	1.15	2.32	1.98	1.95	3.92	2.35	2.33	4.68	2.72	2.72	5.44
pe_0/mul1	0.50	0.40	0.89	1.36	1.27	2.63	2.08	2.05	4.13	2.45	2.43	4.87
pe_0/mul2	1.18	1.16	2.33	1.98	1.95	3.93	2.38	2.37	4.75	2.77	2.77	5.54
pe_0/mul3	0.46	0.37	0.83	1.30	1.23	2.53	1.99	1.99	3.97	2.35	2.36	4.70
pe_0/ADD1	0.02	0.02	0.04	0.04	0.03	0.08	0.06	0.05	0.11	0.08	0.07	0.14
pe_0/ADD3	0.00	0.00	0.00	0.01	0.01	0.01	0.03	0.03	0.05	0.04	0.03	0.07
pe_0/ADD0	0.03	0.12	0.15	0.06	0.24	0.30	0.09	0.35	0.44	0.12	0.46	0.57
pe_0/ADD2	0.00	0.00	0.00	0.02	0.07	0.10	0.06	0.22	0.28	0.08	0.29	0.37
pe_0/ADD4	0.01	0.03	0.04	0.04	0.11	0.14	0.07	0.20	0.27	0.09	0.26	0.35
pe_0/ADD5	0.01	0.03	0.05	0.04	0.10	0.14	0.06	0.17	0.23	0.08	0.22	0.30
pe_1/mul0	1.22	1.20	2.42	2.06	2.02	4.08	2.45	2.44	4.89	2.84	2.84	5.67
pe_1/mul1	0.65	0.59	1.24	1.66	1.63	3.29	2.15	2.16	4.31	2.49	2.53	5.02
pe_1/mul2	1.15	1.15	2.31	1.96	1.97	3.93	2.37	2.42	4.79	2.75	2.82	5.58
pe_1/mul3	0.64	0.59	1.23	1.62	1.62	3.23	2.07	2.11	4.19	2.42	2.48	4.90
pe_1/ADD1	0.02	0.02	0.04	0.04	0.03	0.07	0.05	0.05	0.10	0.07	0.06	0.13
pe_1/ADD3	0.00	0.00	0.01	0.02	0.02	0.04	0.04	0.04	0.08	0.06	0.05	0.11
pe_1/ADD0	0.03	0.12	0.15	0.06	0.23	0.29	0.10	0.34	0.44	0.12	0.45	0.57
pe_1/ADD2	0.01	0.03	0.04	0.04	0.15	0.20	0.08	0.29	0.37	0.11	0.38	0.49
pe_1/ADD4	0.02	0.05	0.07	0.05	0.16	0.21	0.08	0.24	0.32	0.11	0.31	0.42
pe_1/ADD5	0.02	0.05	0.07	0.05	0.14	0.19	0.07	0.21	0.28	0.09	0.27	0.36
pe_2/mul0	1.17	1.16	2.33	1.94	1.93	3.86	2.29	2.30	4.59	2.64	2.67	5.31
pe_2/mul1	0.97	0.91	1.88	1.83	1.74	3.57	2.25	2.15	4.39	2.59	2.47	5.07
pe_2/mul2	1.13	1.09	2.22	1.92	1.85	3.76	2.30	2.23	4.53	2.66	2.60	5.26
pe_2/mul3	0.82	0.81	1.62	1.70	1.64	3.35	2.24	2.14	4.38	2.57	2.45	5.02
pe_2/ADD1	0.02	0.02	0.04	0.04	0.04	0.07	0.05	0.05	0.10	0.07	0.07	0.14

pe_2/ADD3	0.01	0.01	0.02	0.03	0.03	0.06	0.05	0.05	0.10	0.06	0.06	0.13
pe_2/ADD0	0.03	0.11	0.14	0.06	0.22	0.28	0.09	0.32	0.41	0.12	0.42	0.54
pe_2/ADD2	0.02	0.07	0.10	0.05	0.19	0.24	0.08	0.31	0.40	0.11	0.41	0.52
pe_2/ADD4	0.03	0.07	0.10	0.06	0.17	0.23	0.08	0.25	0.33	0.10	0.32	0.43
pe_2/ADD5	0.03	0.07	0.09	0.06	0.15	0.20	0.08	0.21	0.29	0.10	0.27	0.37
pe_3/mul0	1.10	1.14	2.23	1.83	1.89	3.72	2.16	2.26	4.43	2.50	2.63	5.12
pe_3/mul1	0.93	0.88	1.81	1.72	1.64	3.36	2.12	2.07	4.19	2.44	2.39	4.83
pe_3/mul2	1.16	1.14	2.30	1.93	1.92	3.85	2.35	2.35	4.70	2.73	2.75	5.47
pe_3/mul3	0.95	0.88	1.83	1.81	1.69	3.50	2.22	2.11	4.34	2.57	2.46	5.02
pe_3/ADD1	0.02	0.02	0.04	0.04	0.04	0.07	0.05	0.06	0.11	0.07	0.07	0.14
pe_3/ADD3	0.01	0.01	0.02	0.03	0.03	0.06	0.05	0.05	0.10	0.06	0.06	0.12
pe_3/ADD0	0.03	0.12	0.15	0.06	0.23	0.29	0.09	0.34	0.43	0.11	0.45	0.56
pe_3/ADD2	0.02	0.08	0.10	0.05	0.19	0.24	0.08	0.31	0.39	0.10	0.41	0.51
pe_3/ADD4	0.03	0.07	0.10	0.06	0.17	0.23	0.08	0.25	0.33	0.11	0.32	0.43
pe_3/ADD5	0.03	0.07	0.10	0.05	0.16	0.21	0.07	0.22	0.30	0.10	0.29	0.38

Based on the power data, the design power is scalable with respect to the number of active PEs. Figure 63 shows the average power vs. the number of PEs for the following cases: zero-PE (idle), one-PE, two-PEs, three-PEs and four-PEs. The power increment when adding an active PE is roughly the same across the cases. For example, transitioning from idle to 1-PE causes a power jump of 48mW, whereas transitioning from two-PE to three-PE adds 32 mW. The main reason for this behavior is that not all component powers scale linearly with number of PEs.

Figure 64 shows the power increment for the main design components vs. the number of active PEs. All PEs contribute to power dissipation in one-PE, two-PEs, three-PEs and four-PEs. If number of active PEs is set to one-PE, for example, all PEs contribute to the FFT execution, but the equivalent performance is one PE performance. Engaging the PEs in the execution simplifies the control logic and prevents hot spots.

Figure 64 shows that most of the components scale roughly linearity compared with the number of PEs, (e.g., the PEs and RAMs.) However, there are some components with non-linear behavior, (e.g., ROMs and extMemCtl.) Despite the non-linearity, the linear component power dominates, as shown in the 3-dimensional plot in Figure 65. In this figure, the axis that goes through the page is the number of active PEs. Consequently, the total power can be approximated as:

$$\text{Total Power}_{(mW)} = \text{Number\_Of\_Active\_PEs} * 40\ \text{mW}$$

The next observation is that circuits that are not doing useful work are kept to minimal activities. For example, the register block circuit is not dissipating dynamic power, since it is not active during the FFT processing. This is due to the clock gating techniques implemented in the design, which is one of the most useful flip-flop power reduction techniques [24]. Moreover, the leakage power (not shown in Table 48) is 0.914 mW, which is negligible compared with the dynamic power. Finally, the switch power is roughly 2% of the total power.

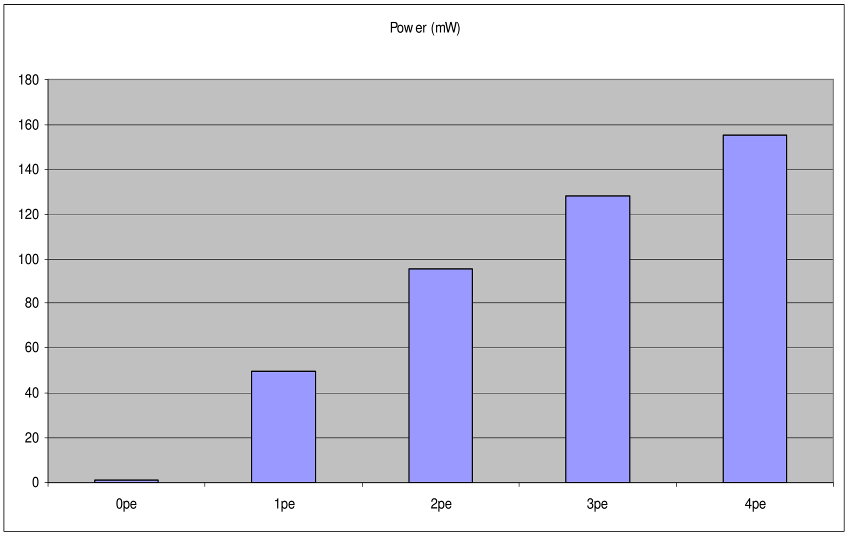


Figure 63. Average power vs. Number of Active PEs

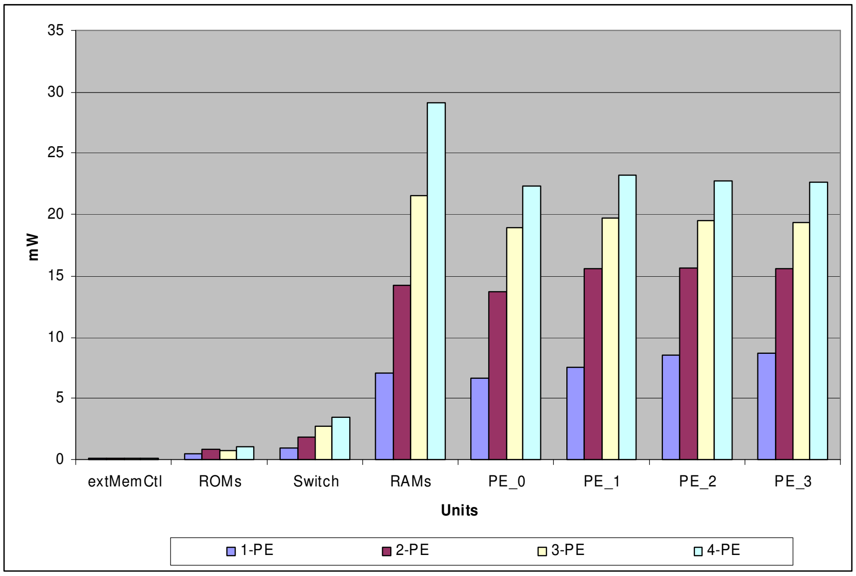


Figure 64. Component Power Across Tests

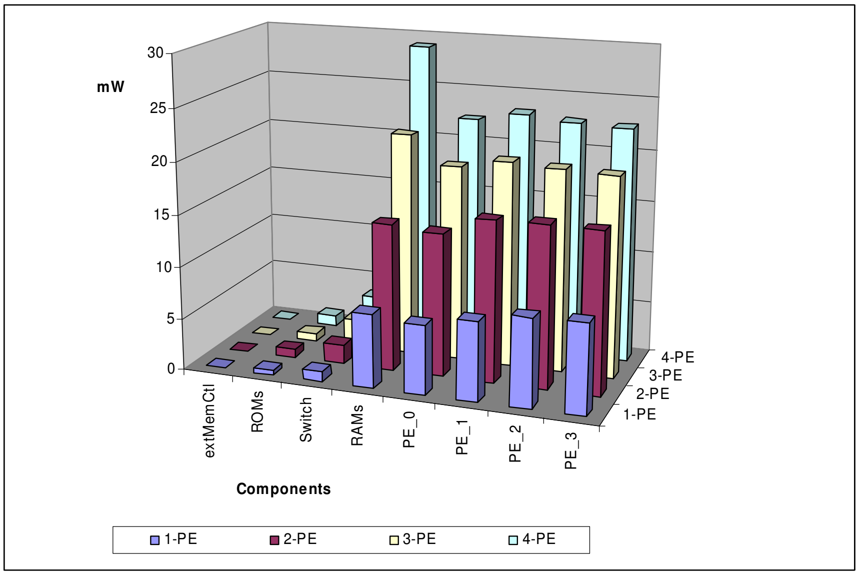


Figure 65. Component Power Across Tests

6.4 Summary

This chapter presented the results and analysis performed on the RTL design. First, the RTL simulation results were presented through a series of frequency spectrum plots. Next, the timing analysis of the design was examined in detail. Finally, a power analysis for the gate-level design was discussed. A summary of the post synthesis results is shown in Table 49.

Table 49. Post Synthesis Results Summary

Category	Parameter	Parameter
Timing	Cycle Time	5.0 ns
	Frequency	200 MHz
Power	Leakage Power	0.9 mW
	Idle Power	1.2 mW
	Max Power	155 mW
	Power Equation	40 mW * Number_Of_Active_PE
Switch Overhead	Switch Power Overhead	2%
	Switch Timing Overhead	18%

Chapter 7

Physical Design

The physical design phase transforms the synthesized netlist into the layout design through a series of floorplan, place and route steps. Once completed, timing and power analysis are conducted on the layout design much like the post-synthesis analysis. Usually, the layout analysis differs from the post-synthesis analysis for mainly three reasons. First, layout analysis is based on extracted Resistance-and-Capacitances (RCs) instead of wireload models. Extracted RCs are more accurate. Secondly, the physical design modifies the synthesized gate netlist by upsizing and downsizing gates. It also adds inverters and buffers to buffer long wire nets. Furthermore, in some cases the cone of the logic is modified to meet area/timing goals. Lastly, the clock tree is present in the physical design and is not assumed to be ideal as in the post-synthesis.

The remaining of this chapter describes the processor physical design and layout timing and power analysis. Lastly, the processor is compared to other published processors.

7.1 Physical Design

Floorplanning is the first step in physical design. It includes defining the core dimensions (height and width), assigning the pins to their locations and placing large macros. The main challenge in placing the macros is to balance the timing, power and area tradeoffs. Several floorplans were examined; however, most of them did not address the main issue for this design, which is to place the macros at nearly equal distances from the switch and the PEs. Figure 66 shows the final version of the floorplan, which offered the best solution. The main features of the floorplan are:

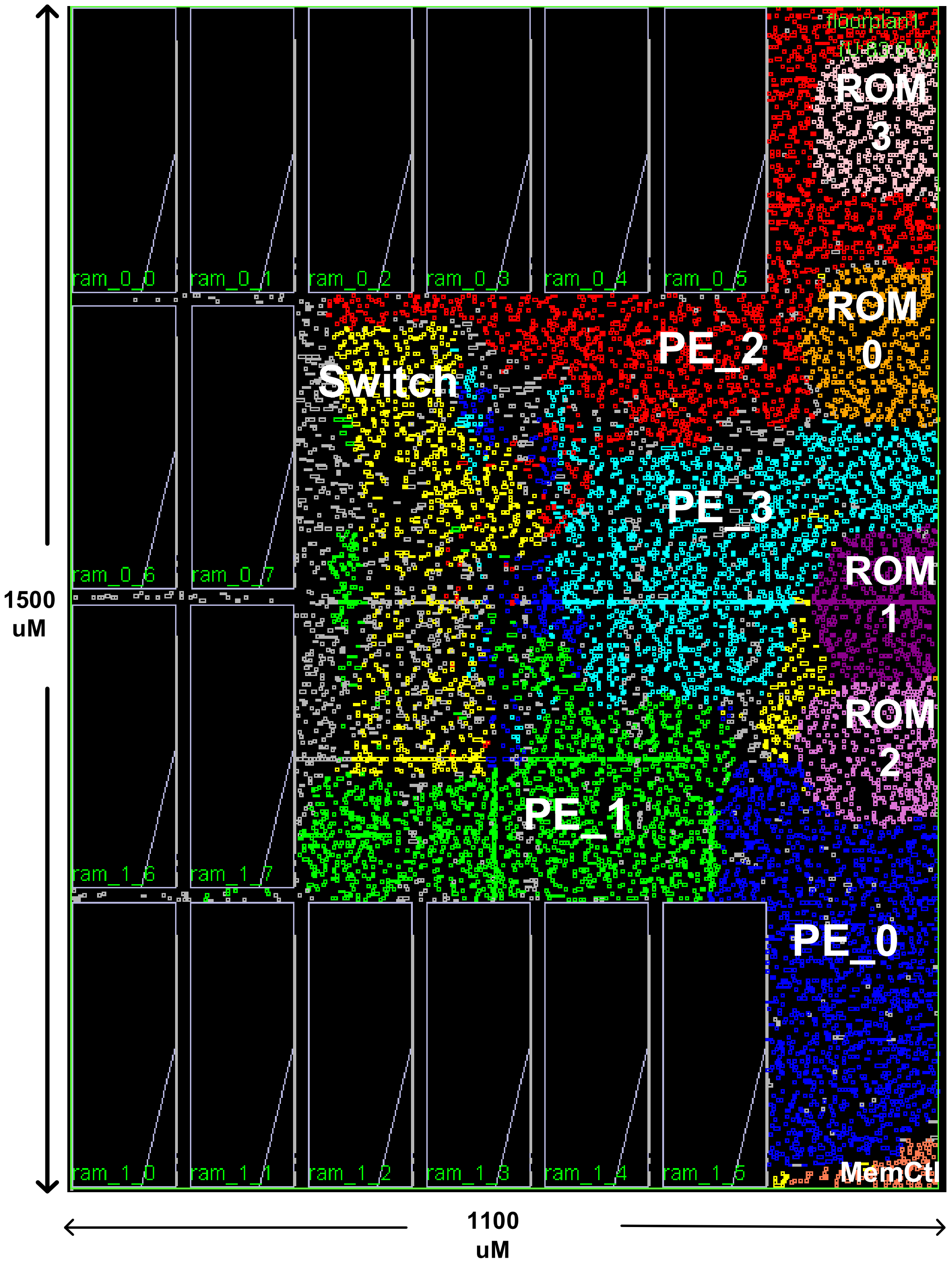


Figure 66. Floorplan

- The RAMs are placed in four major rows. The top and bottom rows have six RAMs each. The two middle rows have two RAMs each. Set_0 RAMs are in the first and second row, while set_1 RAMs are situated in the third and fourth rows.
- Pins are placed in the right side of the core.
- Sandwiched between the RAM rows is the switch component. To the right of the switch are the PE regions. The ROMs and external memory controller are placed on the right edge of the core (from top to bottom). This placement offers roughly equal distance between the RAMs and the critical path components (i.e., the switch and PEs.) The components with a more relaxed timing (i.e., the ROMs and the external memory controller) are pushed out to the right edge of the core.
- 20-µm routing channels are designated around the RAMs. The channels are provided to route data and the address buses.

Because of their large sizes, the RAMs determine the size of the core. The height of the core should fit four stacked RAMs, separated by a 20 µm routing channel. Since the height of the RAM cell is 360 µm, therefore:

The core height (H) = 4*(360 µm) + 3*(20 µm) = 1500 µm = 1.5 mm

The width of the core should fit six RAMs, a 20 µm routing channel and a 220 µm standard cell row. Since the RAM width is 130 µm, therefore:

- The core width (W) = 6*(130 µm) + 5*(20 µm) + 220 µm = 1100 µm = 1.1 mm,
- The core area = H * W = 1.1mm * 1.5mm = 1.65 $mm^{2,}$
- The core aspect ratio = W/H = 0.73.

The design has 85.5K gates which occupies an area of 1.38 mm^2. Hence, the area utilization is 1.38/1.65=0.84. Table 50 illustrates cell statistics. Notice that the number of inverter cells is high. This is expected for this high connectivity design since inverters are commonly used to buffer long wire nets.

Table 50. Cell Statistics

Cell Type	Cell Type	Count	Area
Standard Cells	Inverters	19723	0.080 mm^2
	Buffers	868	0.004 mm^2
	Boolean cells	64878	0.548 mm^2
	Flip-Flops	108	0.003 mm^2
	Standard cell total	85581	0.635 mm^2
Macros	Hard Macros	16	0.748 mm^2
Total		**85597**	**1.383 mm^2**

The completed routed layout design is shown in Figure 67. The design uses six metal layers. The ground and power for the rows of the standard cell are routed in metal-2. The total wire length for the design is just above six meters. The breakdown of the individual layers wire length is illustrated in Table 51. The second and third columns provide the layer length (in meters) and the percentage of the total length. The fourth column shows the number of wires per layer. The last column computes the length/wire ratio for the layer. Since metal-1 and metal-2 have the most route blockages, their total lengths are less than the other layers. Furthermore, since higher layers are used mostly for global and longer nets, their length/wire ratios are higher.

Table 51. Wire Statistics

Layer	Length	% of Total length	Number of Wires	Length/Wire
Metal-1	0.23 m	3.7%	67145	3.37 μm /wire
Metal-2	0.99 m	16.1%	611977	1.47 μm /wire
Metal-3	2.05 m	33.3%	254643	8.04 μm /wire
Metal-4	1.51 m	24.6%	155717	9.68 μm /wire
Metal-5	1.00 m	16.3%	69918	14.32 μm /wire
Metal-6	0.37 m	6.0%	20902	17.61 μm /wire
Total length	6.14 m	100%	1180302	5.20 μm /wire

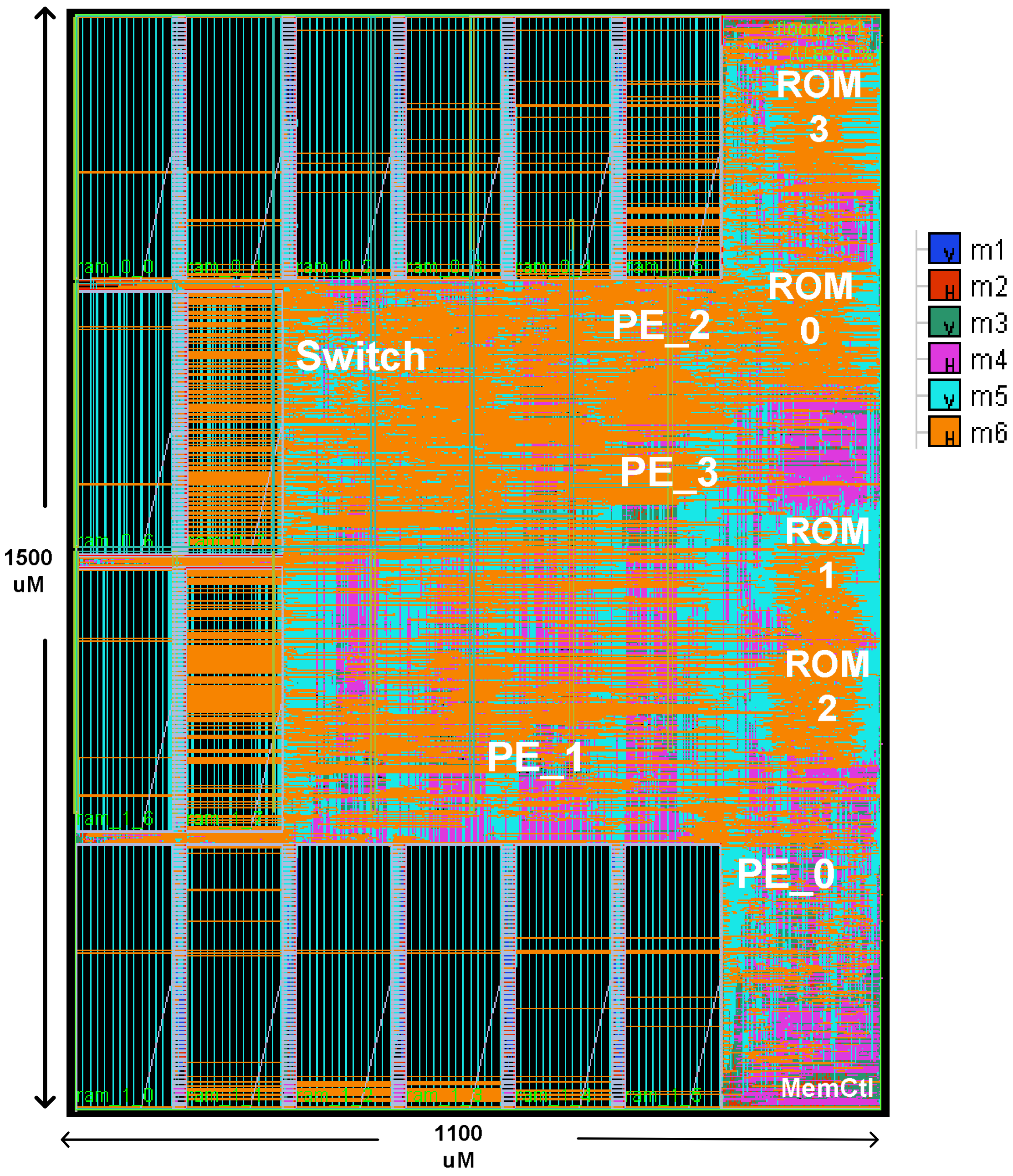


Figure 67. Layout View

Table 52 provides the statistics for the design nets. The data indicates that the design has high connectivity, which is typical for this type of design. This is shown in the relatively high number of pins per net and the average net wire length of 65 µm. Additionally, most of the gates in the design are complex, since the average pins per standard cell is 3.6.

Table 52. Nets and Pins Statistics

Net/Wire Type	Count
Number of nets	93916
Number of cell pins	302175
Average nets per standard cell	1.10
Average pins per standard cell	3.56
Average pins per net	3.25
Average net wire length	65 µm
Average wires per net	12.6 wires

7.2 Timing Analysis

Similar to the post-synthesis timing analysis, the main critical timing path in the layout design is the PE path, which is shown in Figure 60. The path track on the layout is illustrated in Figure 68. It starts and ends with two RAMs in set_1, located at the bottom of the core. The path travels from the source RAM to the switch logic, to PE_0, back to the switch and then finally to the destination RAM. The path delays are shown in Table 53 (as absolute values) and Table 54 (as percentages of total delay). The first column lists the component name, the second column shows post-synthesis timing and the third column gives the post-layout delays. The data shows that:

- Post layout timing is worse by 6.4%. The critical path length has increased to 5.322 ns. Consequently, the clock period is now 5.322 ns and the processor frequency is 188 MHz.
- The overhead of the switch delay is 20%, which is close to the 18% obtained in post-synthesis timing. With 64% of the timing path in its logic, the dominant

component in the critical path is the PE delay. On the other hand, the impact of the switch architecture on timing is one-fifth of the cycle time. This indicates that when increasing the number of PEs, the switch is not likely to be the gating factor for timing.

- The clock skew is 40 ps.

The external memory path increased to 3 ns (from 2.5 ns in post-synthesis.) Despite the increase, the path does not impact the clock frequency. The core inputs should arrive 3 ns before the rise of the clock. This leaves the outside system with 2 ns, which is an adequate time to drive the inputs.

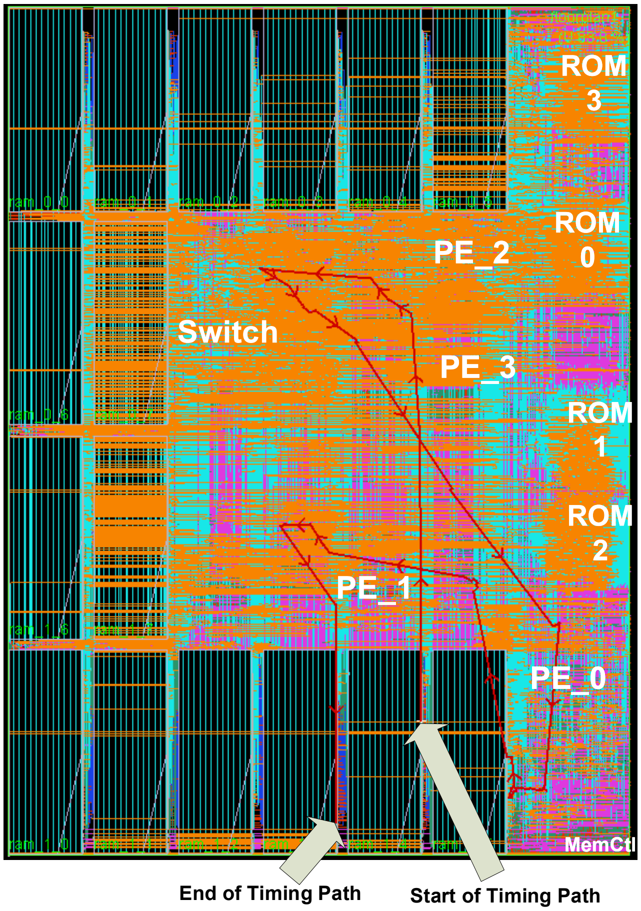


Figure 68. PE Timing Path

Table 53. Timing Delays in the PE Timing Path

Operation	Synthesis Delays	Layout Delays
Drive RAM Data	400 ps	690 ps
Switch Delay 1	530 ps	560 ps
32-bit Subtractor	890 ps	890 ps
32-bit Multiplier	1880 ps	1780 ps
32-bit Adder	830 ps	760 ps
Switch Delay 2	370 ps	500 ps
RAM setup	100 ps	100 ps
Clock Skew	0 (Ideal Clock)	40 ps
Total Path	5000 ps	5320 ps

Table 54. Delay Percentages for the PE Timing Path

Operation	Synthesis Delays	Layout Delays
Drive RAM Data	8%	13%
Switch Delay 1	11%	11%
32-bit Subtractor	18%	17%
32-bit Multiplier	38%	33%
32-bit Adder	17%	14%
Switch Delay 2	7%	9%
RAM Setup	2%	2%
Clock Skew	0%	1%
Total	100%	100%

7.3 Power Analysis

The layout power analysis process is similar to the post-synthesis power analysis. However, the analysis uses the layout netlist and extracted RCs instead. The test signals are the same ones as used in the post-synthesis power analysis. Initially the layout netlist is simulated to capture waveform activity. Next, the extracted RC, the netlist and the waveform activity are simulated in the power estimation tool. Finally, the power reports are post processed to generate the power reports. The power analysis results are summarized in Table 55. The table is organized similarly to the post-synthesis power table.

Table 55. Layout Power Results

Component	1-PE			2-PE			3-PE			4-PE		
	I	S	T	I	S	T	I	S	T	I	S	T
switch	1.44	2.08	3.52	2.46	4.14	6.60	3.48	6.21	9.69	4.49	8.28	12.77
extMemCtl	0.05	0.00	0.05	0.05	0.00	0.05	0.05	0.00	0.05	0.05	0.00	0.05
registers	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.00	0.01
rom0	0.02	0.06	0.08	0.04	0.12	0.16	0.06	0.18	0.24	0.08	0.24	0.33
rom1	0.02	0.04	0.06	0.04	0.08	0.12	0.05	0.12	0.17	0.07	0.16	0.23
rom2	0.02	0.04	0.06	0.03	0.09	0.12	0.05	0.13	0.18	0.07	0.17	0.23
rom3	0.02	0.07	0.09	0.05	0.14	0.19	0.07	0.21	0.28	0.09	0.28	0.37
ram_0_0	0.88	0.04	0.92	1.75	0.08	1.84	2.66	0.12	2.78	3.59	0.17	3.75
ram_0_1	0.88	0.04	0.92	1.75	0.08	1.83	2.66	0.11	2.77	3.59	0.15	3.74
ram_0_2	0.88	0.04	0.92	1.75	0.09	1.84	2.66	0.13	2.79	3.59	0.18	3.76
ram_0_3	0.88	0.05	0.92	1.75	0.09	1.84	2.66	0.14	2.79	3.59	0.18	3.77
ram_0_4	0.88	0.06	0.93	1.75	0.11	1.87	2.66	0.17	2.83	3.59	0.23	3.81
ram_0_5	0.88	0.06	0.94	1.75	0.13	1.88	2.66	0.19	2.85	3.59	0.26	3.84
ram_0_6	0.88	0.04	0.92	1.75	0.08	1.83	2.66	0.12	2.77	3.59	0.16	3.74
ram_0_7	0.88	0.03	0.91	1.75	0.06	1.81	2.66	0.09	2.74	3.59	0.11	3.70
pe_0/mul0	0.64	0.44	1.08	1.24	0.87	2.12	1.85	1.31	3.16	2.45	1.74	4.20
pe_0/mul1	0.50	0.31	0.81	0.96	0.62	1.58	1.43	0.93	2.36	1.89	1.24	3.13
pe_0/mul2	0.53	0.36	0.89	1.03	0.72	1.76	1.54	1.09	2.62	2.04	1.45	3.49
pe_0/mul3	0.54	0.35	0.88	1.04	0.69	1.73	1.54	1.04	2.57	2.04	1.38	3.42
pe_0/ADD1	0.02	0.03	0.05	0.04	0.06	0.10	0.06	0.08	0.15	0.08	0.11	0.19
pe_0/ADD3	0.02	0.02	0.04	0.04	0.05	0.08	0.05	0.07	0.12	0.07	0.09	0.17
pe_0/ADD0	0.06	0.12	0.18	0.11	0.25	0.36	0.16	0.37	0.53	0.21	0.50	0.71
pe_0/ADD2	0.03	0.06	0.10	0.06	0.13	0.19	0.09	0.19	0.27	0.11	0.25	0.36
pe_0/ADD4	0.04	0.07	0.12	0.08	0.15	0.23	0.12	0.22	0.34	0.16	0.29	0.45
pe_0/ADD5	0.04	0.06	0.10	0.07	0.13	0.20	0.11	0.19	0.30	0.14	0.25	0.39
pe_1/mul0	0.50	0.32	0.82	0.97	0.65	1.62	1.45	0.97	2.42	1.92	1.30	3.22
pe_1/mul1	0.51	0.29	0.80	0.99	0.59	1.58	1.47	0.88	2.35	1.95	1.17	3.13
pe_1/mul2	0.54	0.37	0.90	1.04	0.73	1.78	1.55	1.10	2.65	2.06	1.47	3.52
pe_1/mul3	0.57	0.34	0.92	1.12	0.69	1.80	1.66	1.03	2.68	2.20	1.37	3.57
pe_1/ADD1	0.02	0.03	0.05	0.04	0.05	0.09	0.06	0.08	0.13	0.08	0.10	0.18
pe_1/ADD3	0.02	0.02	0.04	0.03	0.04	0.07	0.05	0.06	0.11	0.06	0.08	0.14
pe_1/ADD0	0.06	0.12	0.19	0.11	0.25	0.36	0.17	0.37	0.54	0.22	0.50	0.72
pe_1/ADD2	0.04	0.07	0.10	0.07	0.13	0.20	0.10	0.20	0.30	0.14	0.27	0.40
pe_1/ADD4	0.04	0.05	0.09	0.07	0.11	0.18	0.11	0.16	0.27	0.14	0.22	0.36
pe_1/ADD5	0.04	0.05	0.09	0.07	0.10	0.17	0.10	0.15	0.26	0.14	0.20	0.34
pe_2/mul0	0.60	0.42	1.02	1.17	0.83	2.00	1.74	1.25	2.99	2.31	1.66	3.98
pe_2/mul1	0.54	0.38	0.92	1.05	0.76	1.82	1.56	1.15	2.71	2.07	1.53	3.60
pe_2/mul2	0.52	0.36	0.88	1.02	0.71	1.73	1.52	1.07	2.59	2.02	1.42	3.44

pe_2/mul3	0.56	0.40	0.96	1.09	0.80	1.89	1.62	1.19	2.81	2.15	1.59	3.74
pe_2/ADD1	0.02	0.03	0.05	0.04	0.05	0.09	0.06	0.08	0.14	0.08	0.10	0.18
pe_2/ADD3	0.02	0.03	0.05	0.04	0.05	0.10	0.06	0.08	0.15	0.08	0.11	0.19
pe_2/ADD0	0.06	0.13	0.19	0.11	0.26	0.38	0.16	0.40	0.56	0.22	0.53	0.75
pe_2/ADD2	0.05	0.11	0.16	0.09	0.22	0.31	0.13	0.33	0.46	0.17	0.43	0.61
pe_2/ADD4	0.05	0.09	0.14	0.10	0.18	0.28	0.14	0.28	0.42	0.19	0.37	0.56
pe_2/ADD5	0.05	0.08	0.13	0.09	0.17	0.25	0.13	0.25	0.38	0.17	0.33	0.50
pe_3/mul0	0.54	0.32	0.86	1.05	0.64	1.68	1.56	0.96	2.51	2.07	1.28	3.34
pe_3/mul1	0.50	0.34	0.84	0.98	0.67	1.65	1.46	1.01	2.46	1.94	1.34	3.28
pe_3/mul2	0.51	0.32	0.83	0.99	0.65	1.63	1.46	0.97	2.43	1.94	1.29	3.24
pe_3/mul3	0.50	0.31	0.81	0.98	0.62	1.60	1.45	0.93	2.39	1.93	1.24	3.17
pe_3/ADD1	0.02	0.02	0.05	0.04	0.05	0.09	0.06	0.07	0.13	0.08	0.09	0.18
pe_3/ADD3	0.02	0.02	0.04	0.03	0.04	0.08	0.05	0.06	0.11	0.07	0.09	0.15
pe_3/ADD0	0.04	0.07	0.11	0.08	0.13	0.21	0.11	0.20	0.32	0.15	0.27	0.42
pe_3/ADD2	0.05	0.10	0.15	0.09	0.20	0.29	0.13	0.30	0.43	0.16	0.40	0.57
pe_3/ADD4	0.04	0.06	0.10	0.07	0.12	0.20	0.11	0.19	0.29	0.14	0.25	0.39
pe_3/ADD5	0.04	0.07	0.11	0.08	0.14	0.22	0.12	0.21	0.33	0.16	0.28	0.44
Clk	0.19	1.28	1.47	0.19	1.28	1.47	0.19	1.28	1.47	0.19	1.28	1.47
Buffer	0.51	1.63	2.14	0.89	3.26	4.15	1.26	4.90	6.15	1.63	6.53	8.16

Analyzing the table data shows that:

- The switch power dissipation is 11% of the total power. Hence, the switch does not have a significant power overhead.
- The layout power analysis includes two new components: clock power and buffer power. The clock power represents the clock tree power dissipation. The buffer component represents the buffers and invertors added to the buffer long wires in the layout.
- Figure 69 and Figure 70 show the total power and component power vs. the number of active PEs. Figure 69 demonstrates that the total power trends linearly with number of active PEs. Figure 70 shows that PE and RAM powers exhibit a good linear behavior with almost identical slopes. The switch and buffer components have a linear behavior with lower slopes. However, the clock and external memory controller exhibit non-linear behavior. Fortunately, their contributions are small compared to other linear components. In fact, the total power can be approximated as follows:

Total Power = 30 mW * Number_Of_Active_PEs

- The linearity of the layout power is much better than synthesis power. This is the result of careful floorplanning and placement. By localizing the component gates close to each other, the net wires are shorter and therefore the switching capacitance is reduced.
- The leakage is 1.3 mW, which is higher than post-synthesis leakage (i.e., 0.914 mW.) The increase is attributed to adding extra gates in the layout design (e.g., the clock tree invertors and wire buffers) and upsizing gates.
- The idle power is 3.2 mW. Since the leakage power is 1.3 mW, approximately 1.9 mW of dynamic power is dissipated in the idle state to keep critical state machines alive.

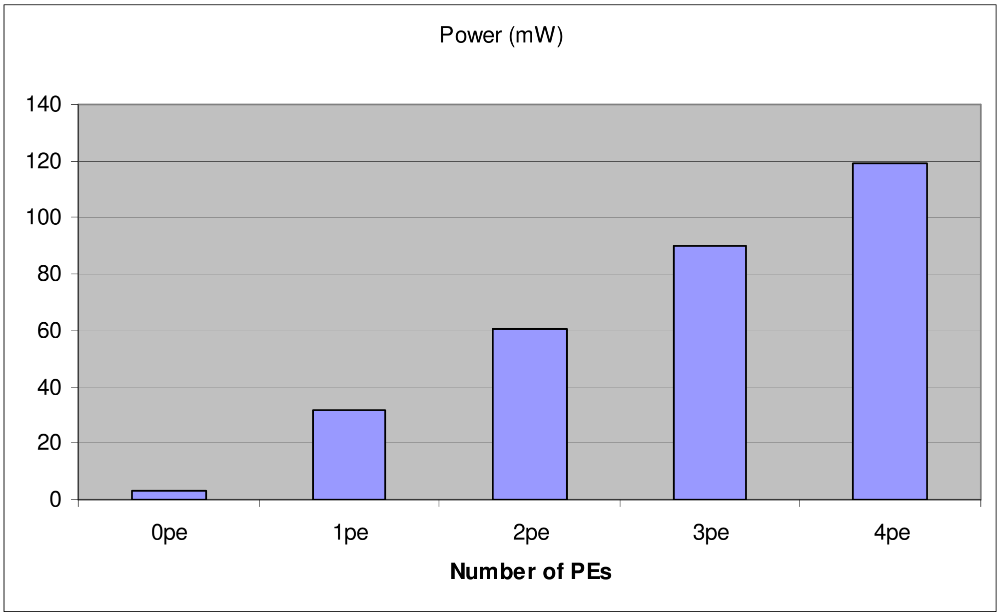


Figure 69. Power vs. Number of Active PEs

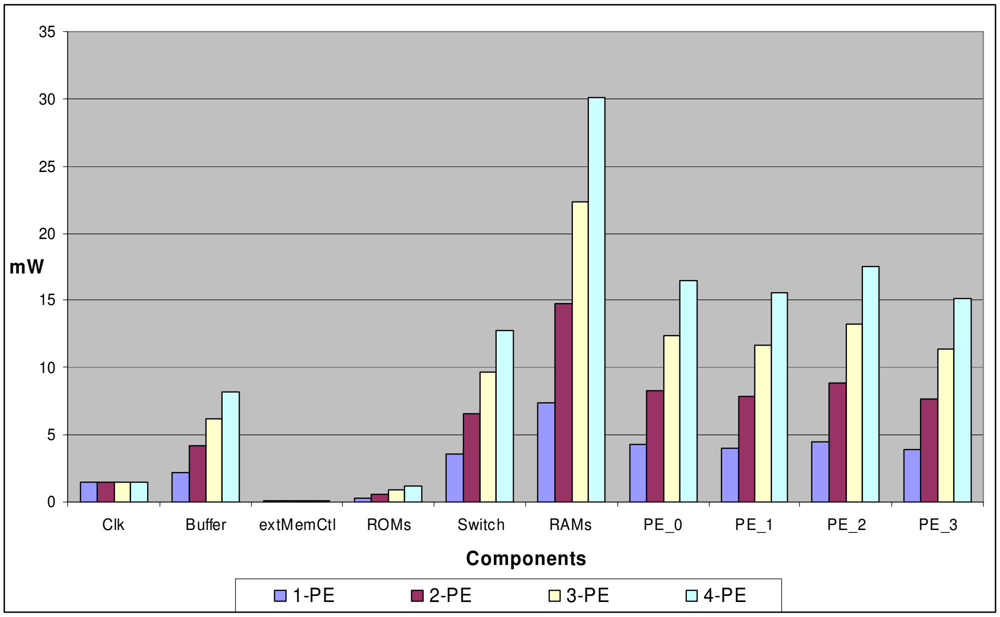


Figure 70. Component Power vs. Number Active PEs

7.4 The Impact of Wire Resistance and Capacitance

This section addresses the following questions. How accurate is the wireload model? What is the impact of extracted RCs on timing and power? The answers to these questions are in Figure 71, which shows the normalized timing and power for the cases:

- Layout with no RCs,
- Post-synthesis timing with WLM,
- Layout timing with RC.

The results are normalized to the layout timing with RC.

As for timing, the data indicates that the WLM is reasonably accurate but is a little optimistic. As a result, the layout timing was off by only 6%. Moreover, the RC constitutes 40% of the critical path. This is not unusual for the 65 nm process.

For power, the wireload model is pessimistic because it has inflated the capacitance of many nodes, which raised the dynamic power by 30% as compared with layout power. Furthermore, 30% of the total power is attributed to wire RCs.

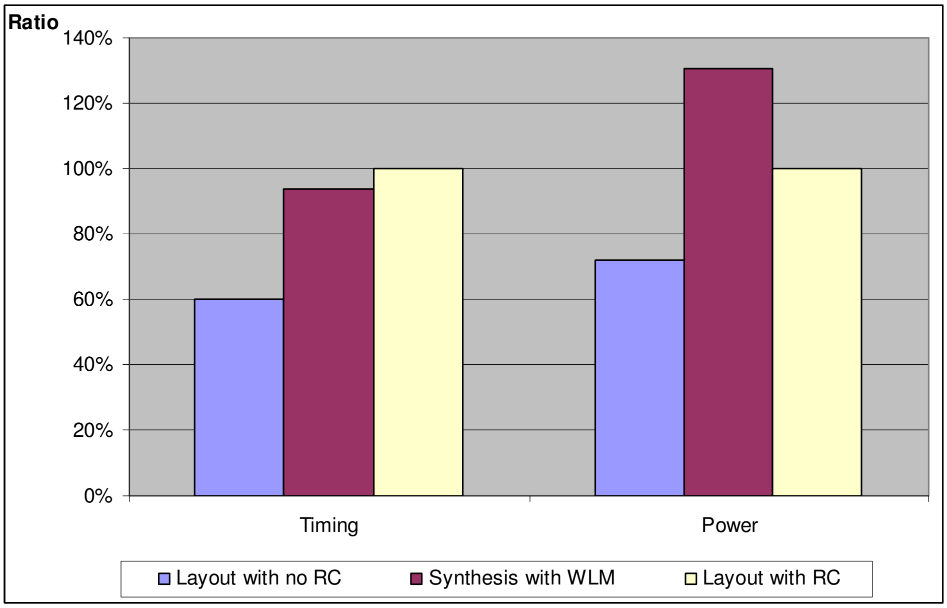


Figure 71. Impact of RC on Power and Timing

7.5 Comparison with other Processors

The performance of the switch-based processor is compared against other processors in a number of important criteria.

The criterion of the total number of cycles to process an *N*-point FFT is one of the most important criteria to compare different architectures [15]. Table 56 and Table 57 illustrate the number of cycles to process 1024-point and 256-point FFTs for a variety of processors. The second column in the tables lists the number of cycles. The normalized number of cycles (with respect to the switch-based processor) is shown in the third column. The data in both tables show the superior performance for the switch-based architecture. Notice that all but the TI DAVINCI processor require more cycles than the switch-based processor.

Table 56. Benchmarks of Fixed-Point Processors for 1024-Point FFT

Processor	Clock cycles	Normalized clock cycles
Switch-based FFT processor	3840	1
Ring-based processor [15].	5280	1.38
Cached FFT [11]	5240	1.36
Texas Instrument TMS320DM644x (DAVINCI) [34]	3878	1.01
Texas Instrument TMS320C64x [32]	6002	1.56
Texas Instruments TMS320C62x [33]	12972	3.38
Analog Devices Sharc processor [36]	9200	2.4

Table 57. Benchmarks of Fixed-Point Processors for 256-Point FFT

Processor	Clock cycles	Normalized clock cycles
Switch-based FFT Processor	768	1
Analog Devices Blackfin Processor [35]	2324	3.03
Texas Instruments C55x [37]	4786	6.23

The rest of the section focuses the comparison between the switch-based processor and ring-processor. The ring processor was chosen for more detailed comparison, since it is a recent research (i.e., published in 2006) and is well documented in [15] and [16]. Moreover, it utilizes four PEs, has similar memory sets and operates at almost the same frequency as the switch-based FFT processor.

An important comparison criterion is the sustained data rate (i.e., the number of samples per second). The “samples per second” is calculated as follows.

$$\begin{aligned} Samples\ \ Per\ \ Second\ (N) &= \frac{N_{\text{samples}}}{Time_{\text{N-Point FFT}}} \\ &= \frac{N_{\text{samples}}}{Cyles_{\text{N-Point FFT}} \times \dfrac{Time}{Cycle}} \\ &= \frac{Frequency\ \times N}{Cyles_{\text{N-Point FFT}}} \end{aligned}$$

Table 58 shows the samples per second for 1024-point FFT for the switch-based and ring processor. The switch-based processor has about 30% higher samples per second rate compared with the ring processor.

Table 58. Samples per Second Criterion

	Samples Per Second (*N*=1024)	Normalized Sample Per Second (*N*=1024)
Switch-Based Processor	50.1 MSPS	100%
Ring-based processor [15]	38.8 MSPS	77%

The final comparison criterion is **"Energy x Time",** since it incorporates energy-efficiency and processor performance [11]. The "Energy x Time" can be expressed in a variety of ways:

$$\begin{aligned}
Energy \times Time &= Power \times [Time\ For\ One\ FFT] \times [Time\ For\ One\ FFT] \\
&= Power \times [Time\ For\ One\ FFT]^2 \\
&= Power \times [Cycles\ Per\ One\ FFT]^2 \times [Clock\ Period]^2 \\
&= Power \times \left[\frac{Cycles\ Per\ One\ FFT}{Frequency}\right]^2
\end{aligned}$$

Because it uses different word sizes and technologies, the ring processor power is scaled prior to the comparison.

To a first order, the main contributors to the power consumption are the PEs and the RAMs. The PE consists mainly of adders and multipliers. The multiplier power scales with M^2, the adder scales with M and the RAM scales with $\log_2 N$; *where M is the data word size and N is the length of the FFT*. Combining these facts with the power analysis, the first order word size (WS) scaling can be expressed as:

$$WS\ Scaling = 0.5 \times \left[\frac{WS\ of\ switch\ processor}{WS\ of\ ring\ processor}\right]^2 + 0.5 \times \frac{WS\ of\ switch\ processor}{WS\ of\ ring\ processor}$$

In [14], Bass utilized the following WS scaling:

$$WS\ Scaling = \frac{1}{3} \times \left[\frac{WS\ of\ Bass\ processor}{WS\ of\ other\ processor}\right]^2 + \frac{2}{3} \times \frac{WS\ of\ Bass\ processor}{WS\ of\ other\ processor}$$

The above two equations are similar but not identical. This highlights that power scaling is a first-order approximation exercise.

The next scaling is power scaling due to different technologies. Technology scaling is extremely complex because of the strong dependencies on the foundry and the process. As a result, the below analysis is presented with and without technology scaling.

Suppose that processor$_x$ uses an x-meter process and VDD$_x$ power supply, while processor$_y$ uses a y-meter process and VDD$_y$ power supply. The dynamic power is proportional to CV^2, and C scales approximately with the transistor dimensions [30]. The power for processor$_y$ is estimated as:

$$Power_y = \left[\frac{VDD_y}{VDD_x}\right]^2 \times \frac{y}{x} \times Power_x$$

Reference [15] uses the same method to scale energy.

Table 59 shows the "Energy x Time" comparison with the scaled-power and the non-scaled power for the ring processor. The energy figures correspond to a 1024-point FFT operation. Compared to non-scaled, the switch-based processor has almost a sixth of the "Energy x Time" of the non-scaled ring processor. Moreover, the switch-based is slightly better than the scaled ring processor. Hence, the switch processor is slightly more energy efficient.

Table 59. Comparing "Energy x Time" Criterion

	Power (mW)	**Scaled Power** (mW)	**Energy x Time** (J x micro-S)	**Energy x Time** (normalized)
Switch-Based Processor	119	119	49.6	100%
Ring Processor (no scaling)	410	410	285.8	576%
Ring Processor (with scaling)	410	77.4	53.9	109%

7.6 Summary

This chapter discussed the physical design of the processor in detail. Post-layout timing and power analysis were examined. Finally, the processor was compared to other published processors. Table 60 presents a data sheet that summarizes the processor physical design features.

Table 60. Data Sheet

Category	Parameter	Parameter
Area	Core Height	1.5 mm
	Core Width	1.1 mm
	Aspect Ratio	0.73
	Area	1.65 mm^2
	Cell Area	1.383 mm^2
	Cell Count	85597
	Utilization	83.8 %
	Total Wire Length	6.138 m
Timing	Cycle Time	5.32 ns
	Frequency	188 MHz
Power	Leakage Power	1.3 mW
	Idle Power	3.2 mW
	Max Power	119 mW
	Power Equation	30 mW * Number_Of_Active_PE
Switch Overhead	Switch Power Overhead	11%
	Switch Timing Overhead	20%
	Switch Area	9% of the core area (excluding RAMs)

Chapter 8

Conclusion

This dissertation has examined the realization of an FFT processor by a switch-based architecture which consists of a switch fabric, M processing elements and $2M$ memories. Each processing element implements a radix-2 butterfly operation. The memory elements are single-port and each has a capacity of $N/(2M)$ entries. In addition, the twiddle factors are stored in M Read Only Memories. The non-pipeline form of the architecture was designed and implemented in 65 nm low power CMOS technology. The timing and power analysis was presented in detail.

If the number of memories is large, the physical design might be impacted. An idea to mitigate this issue is discussed later in this section. The remaining of this chapter describes the key contributions and future research.

8.1 The Key Contributions

The following are the key contributions of this research:

- Developed a switch-based architecture for non-pipeline and pipeline processors. The non-pipeline has a speedup of M compared to a single PE. The pipeline has a speedup of $M\log_2 N$ compared to a single PE. Moreover, the architecture employ single-port RAMs for data storage, which is significantly reduces power (compared with delay elements) and design complexity.
- Developed a memory management algorithm to detect and avoid memory conflicts for M PEs.
- Implemented the non-pipeline form in 65 nm low power CMOS technology. The results showed that switch has a very modest impact on the processor area, timing and power consumption. This suggests that the number of PEs could be increased without significantly impacting the processor timing and power.

- The implementation demonstrated the configurability and power scalability features of the switch-based processor.

8.2 Future Research

In the designs that employ large number of PEs, the number of memories could impact physical design. An idea to mitigate this issue is to utilize wider memories and prefetch/store buffers instead of the regular memories. The wider memory stores *k*-words in each entry, where *k*>2. The prefetch/store buffers are capable of storing *k*-words. The buffers are implemented as flip-flop arrays which are optimized for area and power [24]. A single memory access moves *k*-words between memories and the buffers. The switch accesses the buffers instead of the memories. This scheme reduces the number of memories by a factor equal to the increase in the memory width. The downside is the increase in access latency, which can be mitigated by pipelining techniques.

This work has demonstrated the implementation of a switch-based architecture for one of the most widely used DSP algorithms. One idea is to examine utilizing the architecture in other DSP algorithms. Because of its similar operations to FFT, the Discrete Cosine Transform (DCT) is a prime candidate for future switch-based design.

Additionally, reducing the PE power, specifically the multiplier power, is highly desirable. One approach is to consider a lower datapath width during power-saving modes. Since twiddle factors are fixed values, multipliers with pre-computation techniques ([38] and [39]) could provide potential power savings.

Moreover, the size of the ROM could be examined for further optimization. While in this research the number of ROM entries was reduced; one area of future work is to reduce the width of the ROM word. Examining the tradeoffs between the result accuracy versus ROM word size will lead to more efficient ROM size.

Lastly, the memory size of the pipelined version of the switch-based architecture can be reduced by employing the memory synchronization techniques reported in [9].

Appendix: Wallace Multiplier

The Wallace multiplier was first introduced by Wallace in [40]. The multiplier has been studied and examined in many text books and publications, including [41] and [42]. The Wallace multiplier is a column compression multiplier, with a delay proportional to the logarithm of the operand word length [42]. The multiplier consists of three steps: bit product generation, column compression and carry lookahead adder, as shown in Figure 72. The bit product generation employs N^2 AND gates to form the products.

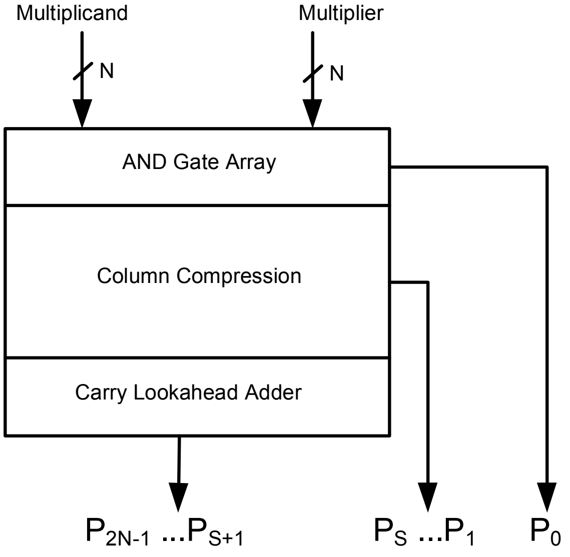


Figure 72. N-bit by N-bit Wallace Multiplier [42]

The column compression compresses the bit product matrix into two rows. Figure 73 illustrates the column compression for a 6-bit by 6-bit multiplier [42]. The compression step processes three rows at a time. Three bit products of the same binary height are combined using a (3,2) counter. If only two bits are present, a (2,2) counter is used to compress the bit products. The reduction process is repeated until there are two rows remaining. The number of steps is of O(log*N*). The final step is adding the last two rows by a carry lookahead adder. In the Wallace multiplier, some of the least significant bits of the product are generated during the compression steps. As a result, the final step adder is smaller than other column compression multipliers (e.g., Dadda's multiplier) [42].

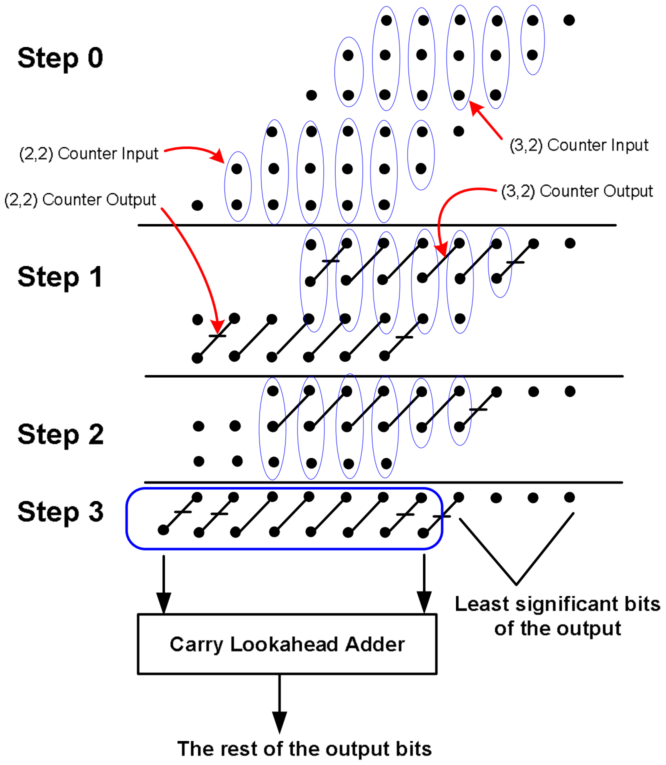


Figure 73. A 6-bit by 6-bit Wallace Multiplier

Bibliography

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.,* vol. 19, pp. 297-301, 1965.

[2] H. L. Groginsky and G. A. Works, "A pipelined fast Fourier transform," *IEEE Transactions* on *Computers,* vol. C-19. pp. 1015-1019, 1970.

[3] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," *Proc. of URSI International Symposium on Signals, Systems, and Electronics*, pp. 257-262, 1998.

[4] S. He and M. Torkelson. "Design and Implementation of a 1024-point Pipeline FFT Processor," *IEEE Custom Integrated Circuits Conference*, pp. 131-134, May 1998.

[5] P. Tsai, T. Lee and T. Chiueh, "Power-Efficient Continuous-Flow Memory-Based FFT Processor for WiMax OFDM Mode," *International Symposium on Intelligent Signal Processing and Communication Systems (IPACS 2006),* pp. 622-625, December 2006.

[6] A. El-Khashab and E. E. Swartzlander, Jr., "The Modular Pipeline Fast Fourier Transform Algorithm and Architecture," *Proc. of the Thirty-Seventh Asilomar Conference on Signals, Systems, and Computers*, pp. 1463-1467, Pacific Grove, CA, November 2003.

[7] A. M. El-Khashab and E. E. Swartzlander, Jr., "A modular pipelined implementation of large fast Fourier transforms," *Proc. of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers*, pp. 995-999, Pacific Grove, CA, November 2002.

[8] A. M. El-Khashab and E. E. Swartzlander, Jr., "An architecture for a radix-4 modular pipeline fast Fourier transform," *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 378-388, June 2003.

[9] A. M. El-Khashab, *Modular Pipeline Fast Fourier Transform Algorithm*, Ph.D. Dissertation, The University of Texas at Austin, 2003.

[10] B.M. Baas, "A generalized cached-FFT algorithm," *IEEE International Conference on Acoustic, Speech and Signal Processing*, Vol. 5, pp. 89-92, March 2005.

[11] B. M. Baas, “A low-power high-performance 1024-point FFT processor,” *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 380-387, March 1999.

[12] B. M. Baas, “An energy efficient single-chip FFT processor,” *Proceedings of the Symposium on VLSI Circuits*, pp. 164-165, June 1996.

[13] B. M. Baas, “A 9.5mW 330 µsec 1024-point FFT processor,” *Proceedings of the Custom Integrated Circuit Conference*, pp. 127-130, 1998.

[14] B. M. Baas, *An Approach to Low-Power, High-Performance, Fast Fourier Transform Processor Design*, Ph.D. Dissertation, Stanford University, 1999.

[15] G. Zhong, F. Xu and A. N. Willson, Jr., “A power-scalable reconfigurable FFT/IFFT IC based on a multi-processor ring,” *IEEE Journal of Solid-State Circuits*, Vol. 41, pp. 483-495, February 2006.

[16] G. Zhong, F. Xu and A. N. Willson , Jr. “An energy-efficient reconfigurable FFT/IFFT processor based on a multi-processor ring,” *XII European Signal Processing Conference (EUSIPCO),* pp. 2023-2026, Vienna, Austria, 2004.

[17] D. Cohen, “Simplified control of FFT hardware,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, pp. 577-579, December 1976.

[18] M. C. Pease, “Organization of large scale Fourier processors,” *JACM*, vol. 16, pp. 474-482, 1969.

[19] L. G. Johnson, “Conflict free memory addressing for dedicated FFT hardware,” *IEEE Transactions on Circuits and Systems, II*, vol. 39, pp. 312-316, 1992.

[20] Y. Ma, “An effective memory addressing scheme for FFT processors,” *IEEE Transactions on Signal Processing*, vol. 47, pp. 907-911, 1999.

[21] Y. Ma and L. Wanhammar, “A hardware efficient control of memory addressing for high-performance FFT processors,” *IEEE Transactions on Signal Processing*, vol. 48, pp. 917-921, 2000.

[22] H. Saleh, B. Mohd, A. Aziz and E. E. Swartzlander, Jr. “Contention-Free Switch-Based Implementation of 1024-point Fourier Transform Engine,” *25th IEEE International Conference on Computer Design*, pp. 7-12, Lake Tahoe, CA, October 2007.

[23] B. J. Mohd, A. Aziz and E. E. Swartzlander, Jr. “The Hazard-Free Superscalar Pipeline Fast Fourier Transform Algorithm and Architecture,” *15th annual IFIP VLSI SoC 2007*, pp. 194-199, Atlanta, October 2007.

[24] B. Mohd, M. Saint-Laurent, P. Bassett, S. Imam "Reducing Flip-Flop Power for DSP Design," *Third Annual Austin Conference on Integrated Systems and Circuits,* pp. 34-39, Austin, TX, May 2008.

[25] J. Shen and M. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*, New York: McGraw-Hill, 2005.

[26] S. Fung, *et al.*, "65nm CMOS High Speed, General Purpose and Low Power Transistor Technology for High Volume Foundry Application," *Symposium on VLSI Technology Digest of Technical Papers***,** pp. 92-93, June 2004.

[27] A. Steegen, *et al.*, "65nm CMOS technology for low power applications," *IEEE International Electron Devices Meeting Technical Digest,* pp. 64-67, December 2005.

[28] R. G. Lyons, *Understanding Digital Signal Processing,* Second Edition, Upper Saddle River, New Jersey: Prentice Hall, 2004.

[29] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing,* Second Edition, Upper Saddle River, New Jersey: Prentice Hall, 1999.

[30] S. Kang and U. Lebleici, *CMOS Digital Integrated Circuits: Analysis and Design,* Third Edition, New York: McGraw Hill, 2003.

[31] N. Weste and D. Harris, *CMOS VLSI Design: A Circuit and Systems Perspective*, Third Edition, Boston: Addison Wesley, 2005.

[32] FFT Benchmarks, Texas Instruments Inc. [Online]. Available: http://www.ti.com/sc/docs/products/dsp/c6000/benchmarks/64x.htm#fft.

[33] FFT Benchmarks, Texas Instruments, [Online]. Available: http://www.ti.com/sc/docs/products/dsp/c6000/benchmarks/62x.htm#fft.

[34] FFT Benchmarks, Texas Instruments Inc [Online]. Available: http://focus.ti.com/lit/ml/sprt379a/sprt379a.pdf.

[35] FFT Benchmarks, Analog Devices, [Online]. Available: http://www.analog.com/processors/blackfin/overview/benchmarks/comparison.html.

[36] SHARC Processor Benchmarks, Analog Devices Inc, [Online]. Available: http://www.analog.com/processors/sharc/overview/benchmarks/index.html.

[37] C55x DSPs Benchmarks, Texas Instruments Inc., [Online]. Available: http://focus.ti.com/dsp/docs/dspplatformscontentaut.tsp?sectionId=2&familyId=325&tabId=508.

[38] K. Muhammad, *Algorithmic and architectural techniques for low power digital design signal processing*, Ph.D. Dissertation, Purdue University, 1999.

[39] H. Choo, K. Muhammad and K. Roy, “Two’s Complement Computation Sharing Multiplier and Its Applications to High Performance DFE,” *IEEE Transactions on Signal Processing*, Vol. 51, pp. 458-469, February 2003.

[40] C. S. Wallace, “A Suggestion for a Fast Mulitplier,” *IEEE Transactions on Electronic Computers*, Vol. EC-13, pp. 14-17, 1964.

[41] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Design*, New York: Oxford University Press, 2000.

[42] L. Dadda “Some Schemes for Parallel Multipliers,” *Alta Frequenza,* Vol. 34, pp. 349-356, 1965.

[43] F. Harris "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform", *Proceedings of the IEEE*, Vol. 66, No.1, pp. 51-83, January 1978.

VITA

Bassam Jamil Mohd (Mohammed) was born on April 24, 1968 in Saudi Arabia, the son of Palestinian immigrants from the West-bank village of Kufr-Rai. After graduating from High School in 1985, he attended King Fahd University at Dhahran, Saudi Arabia. He received a Bachelor of Science degree in Computer Engineering in 1990. He entered Graduate School at the University of Louisiana at Lafayette in 1991. He received a Master of Science degree in Computer Engineering from the University of Louisiana, Lafayette in 1992. Since 1992, he has worked for several semiconductor companies including Intel, Sun and Synopsys. He is currently working for Qualcomm's Austin DSP Design Center. His experience spans circuit design, logic design, verification and power analysis. His research interest includes DSP design and low power design techniques.

Permanent Address: 1706 Willow Bluff Drive, Pflugerville, Texas 78660

This dissertation was typed by the author.