# ABSTRACT

SULE, AMBARISH MUKUND. Design of Pipeline Fast Fourier Transform Processors using 3 Dimensional Integrated Circuit Technology. (Under the direction of Prof. W. Rhett Davis).

Fast Fourier Transform (FFT) processing is an important component of many Digital Signal Processing (DSP) applications and communication systems. We focus on applications requiring large-point FFTs ($> 1024$), and where high-speed of operation has priority. These range from Orthogonal Frequency Division Multiplexing (OFDM) based systems like Terrestrial Digital Video Broadcasting, very high speed digital subscriber loop systems and radar-based detection in military applications. We look at some of the popular FFT Architectures and compare their resource requirements and throughput.

We observe that the performance of pipelined large-point FFTs is limited due to the physical size and access times of the numerous First-in-First-out (FIFO) memories present in the design. For these pipeline FFTs, the critical path delay through arithmetic elements remains constant even as the number of stages increases, but the memory requirement almost doubles with every stage, thus increasing the memory access times in later stages. We demonstrate the advantage of using 3D Integrated Circuit Technology (3DIC) to assist in memory banks interleaving and stacking, leading to a boost in their performance. If the speed requirements for an application are less stringent, the same design could be tweaked and supply voltage lowered to get a low power design.

The 3DIC design methodology we use involves commercial single-tier CAD tools with Perl, Python and TCL scripts to process their inputs and tie-up their single-tier outputs into a multi-tiered format. The design experiments utilize the 3-tier, 3-metal layer, fully depleted Silicon-on-Insulator 3D 0.18 $\mu m$ manufacturing process from MIT Lincoln Labs. Memory interleaving and stacking is demonstrated by the design of a 16 KB FIFO which can operate at 800 MHz. Finally, a design for an 8192-pt. FFT with 24 bits fixed-point inputs (12-bits each of real and imaginary) is proposed which uses the Radix-2/4/8 Multi-path Delay Commutator Pipelined Architecture and which can operate at 214 MHz to give a new FFT set every 19 $\mu s$.

**Design of Pipeline Fast Fourier Transform Processors using 3 Dimensional Integrated Circuit Technology**

by

**Ambarish Mukund Sule**

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

**Computer Engineering**

Raleigh, North Carolina

2007

**Approved By:**

| | |
|---|---|
| Prof. William Rhett Davis<br>Chair of Advisory Committee | Prof. Paul Franzon |
| Prof. Xun Liu | Prof. Steffen Heber |

# Dedication

Dedicated to my parents Mukund and Madhavi Sule

and wife Ratnaprabha

# Biography

Ambarish Mukund Sule received the Bachelor of Engineering (B.E.) Degree in Electronics Engineering from Veermata Jijabai Technological Institute (V.J.T.I.), University of Mumbai, India in 1999 and the M.S. degree in Computer Engineering from the North Carolina State University in 2003. He continued working towards the PhD degree in Computer Engineering at NCSU and defended his dissertation in Dec 2007. From the Fall of 2002 to 2007, he has worked under the guidance of Prof. Rhett Davis in the research areas of ASIC and System-level VLSI Design.

Along the way, Ambarish has worked as a Software Engineer at Infosys Technologies Ltd. Pune, India, and as an ASIC Design and Verification Engineer at Texas Instruments, Bangalore, India and Qualcomm, Cary, North Carolina. He is a member of the Honor Society of $\phi\kappa\phi$ (Phi Kappa Phi) and the Institute of Electrical and Electronics Engineers (IEEE).

## Acknowledgements

First and foremost I would like to thank my parents, wife and sister Anjali for everything they have given me in life. It is only due to their love, support and encouragement that I have achieved the goal of obtaining the PhD degree. I thank my parents Mukund and Madhavi Sule for continuously inspiring me with immense hard work and dedication towards their goals. Their upbringing has laid the all-important foundation for my success in life and motivated me to go for the PhD.

I am happy to have received all the love and support that I needed (and more) from my wife Ratnaprabha in this endeavor. This has been a long and eventful journey and she has been the best companion I could ever ask for. I am thankful for all the selfless sacrifices she has made and her patience throughout my PhD studies. I am privileged to have her by my side.

I thank my parents-in-law Sharadchandra and Sheetal Pradhan for continuing to keep faith in me and supporting my quest for the PhD. I am grateful for their patience and the trust they put in me through all these years. Special thanks to my daughter Tejaswini for spreading so much joy in my life. I thank my cousin Pushkar and sister-in-law Aparna Tamhane for all the help I received from them during my graduate studies in Raleigh.

I sincerely thank my advisor Prof. Rhett Davis for giving me the opportunity to work under his guidance. His vision and ideas are primarily responsible for the research presented in this work. His enthusiasm towards VLSI Design is very inspiring and contagious. I have learned some fantastic things about ASIC Designing and EDA

flows from him which should stand me in good stead in the future. It has been an honor to have been advised by him.

I thank Prof. Paul Franzon, Prof. Xun Liu and Prof. Steffen Heber for agreeing to be on my Advisory Committee. Prof. Franzon has taught me the basics of ASIC and digital design. I will always appreciate the simplicity and efficacy of his teaching methods. Prof. Liu introduced me to the intricacies of EDA tool algorithms which has been very helpful for this research. His course projects were some of the most challenging and fun I have ever done.

I thank the North Carolina State University for providing a great environment for education and research. I had a truly international feel at this university, with the 4 members of my advisory committee originally coming from the different continents of North America, Asia, Australia and Europe.

I am thankful to all my friends in the MUSE research group, Hao Hua, Ravi Jenkal, Samson Melamed, Chris Mineo and Kaushal Modi. They all have contributed to different aspects of this work and made my stay at NCSU enjoyable. The biggest help I received was from Hao, whose 3DIC flow has been the major engine for this research.

Finally, I would like to thank Kim O'Donnell for giving me an opportunity to work at Qualcomm as an ASIC Design and Verification Engineer. This experience has been extremely valuable for my research.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Fast Fourier Transform (FFT) processing is an important aspect of many Digital Signal Processing applications and systems. Real-time or fast execution is an important criteria for many applications. For example, wide-band Orthogonal Frequency Division Modulation (OFDM) systems, biomedical instrumentation and radar usage in military domain are some of the applications requiring high-speed and large-point FFT systems as one of their key components.

FFT architectures with structured pipelines have been commonly used to meet the fast, real-time processing demands. But pipeline FFT processors suffer the problem of being memory-bandwidth limited for large-point FFTs. The main contribution of this work is the demonstration of high-speed large-point FFT designs with the novel use of FIFO memories in 3DIC Technology.

The rest of the report is arranged as follows: This chapter introduces 3D Integrated Manufacturing Technology and the MIT-LL 3DIC manufacturing process.

Chapter 2 introduces the concepts of Discrete Fourier Transforms and Fast Fourier Transform. It reviews the common FFT Algorithms in literature and the hardware Architectures that are commonly used to compute using these algorithms. Chapter 3 introduces the 3DIC flow we use to design and research multi-tier circuits. Chapter 4 demonstrates the 3DIC flow with a simple FIFO design which is partitioned into 3 tiers. Chapter 5 discusses the proposed FFT Architecture in detail. Chapter 6 discusses the estimated performance for the proposed FFT Design.

## 1.1 3DIC Technology

3-Dimensional Integrated Circuit Technology (3DIC) involves the stacking of chips (thus creating multiple tiers) to increase system integration. The different tiers could be manufactured in different processes or could even be analog and digital respectively. One of the goals of 3DIC technology is to increase the number of transistors that can be successfully fabricated on a chip without shrinking the transistors themselves.

Davis et.al. [24] present a general overview of 3DIC technologies and introduce the motivation for moving from conventional (one-tier) to 3DIC Technology.

Figure 1.1 summarizes some of the different 3D interconnect approaches being currently researched.

Wire bonding is one of the most common approaches, where the wires connect the individual die in a stack. In general, connections between chips go through the board or chip carrier and back to other chips in the stack. Wire bonds are usually

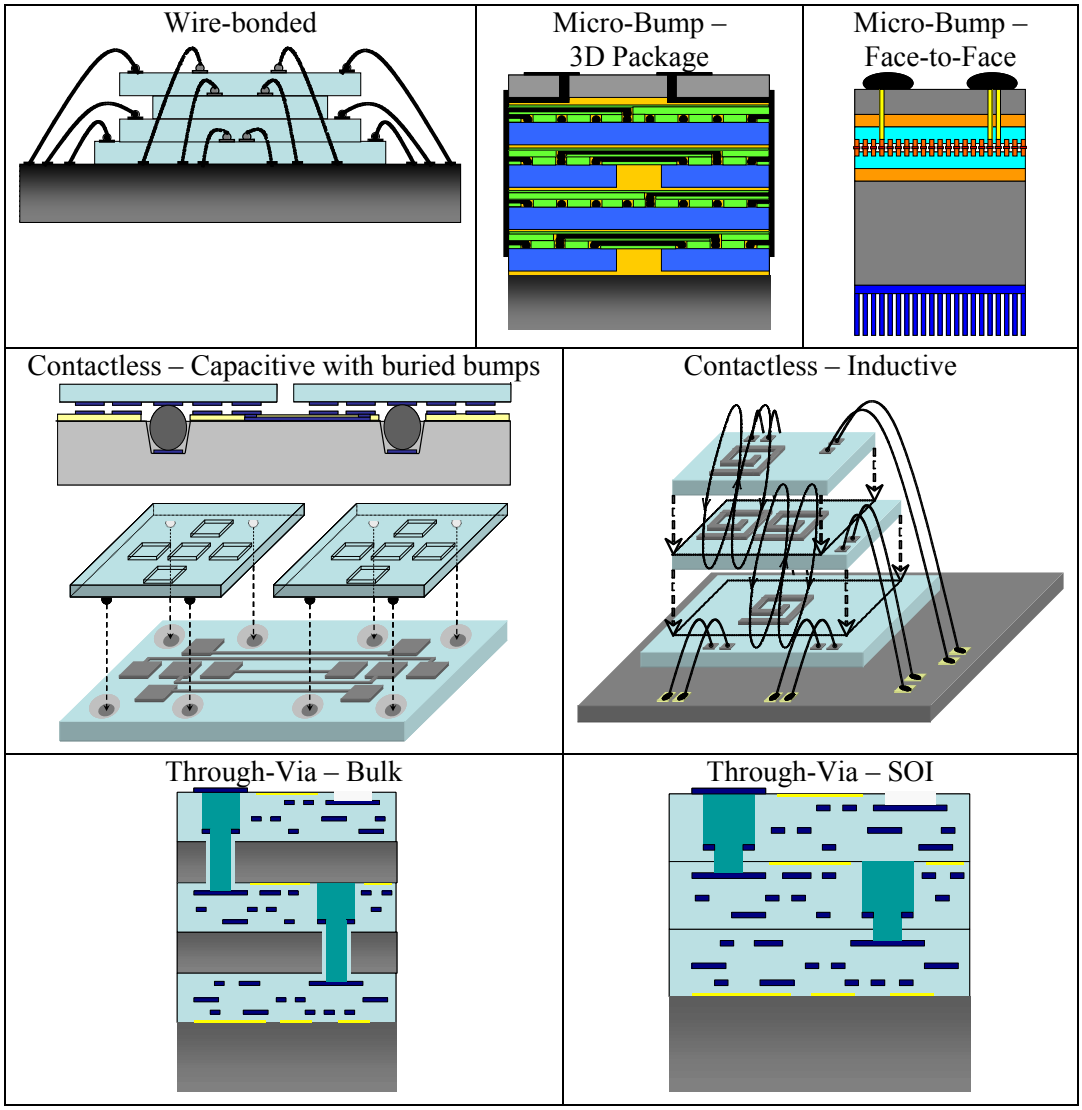


Figure 1.1: Illustration of Vertical Interconnect Technologies [1][24]

possible only on the chip's periphery, which limits interconnect density.

Microbump technology involves the use of solder or gold bumps on the surface of the die to make connections. 3D package technology [49] involves embedding previ-

[1]Thanks to Dr. John Wilson and Prof. Paul Franzon for creating these figures

ously fabricated die into a set of carrier wafers with a fixed size, enabling engineers to assemble them into a tight cube.

Face-to-face microbump technology [14] offers the ability to shorten the wires between tiers and improve performance by reducing parasitics. Taking connections out of the chip stack requires the use of this technology in conjunction with a wire-bonded or through-via approach.

Contactless or AC-coupled interconnection involves the use of capacitive or inductive coupling to communicate between tiers [58]. This approach eliminates the processing steps for creating inter-tier DC connectivity and eliminates the need to route signals to the periphery, allowing for reduced wire lengths.

Capacitive coupling [44, 26] uses half capacitors formed from the top level of metal. The density of these interconnects depends on the distance between the tiers, the rise and fall times of the technology, and the dielectric constant of the gap.

In the case of through-via interconnection, assembly occurs at the wafer level, placing a second wafer face down on the first wafer (face-to-face) and subsequent wafers face down (face-to-back) as the number of tiers grows. The manufacturing process then etches holes through the upper wafer into the lower wafer and fills the holes with tungsten to provide connectivity. As in the 3D package approach, the assembly process in through-via approaches does not limit the number of possible tiers; rather, heat inside the stack is the limiting factor.

Bulk technologies [42] have demonstrated through-via interconnection by first coating the hole with an insulator. Silicon-on-insulator (SOI) technologies [67] avoid

the need for passivating the hole by polishing the substrate away completely, down to the buried oxide. As for routing resources, the through-via approaches consume all layers in the upper tier in addition to the top layer in the lower tier

3DICs are a possible solution for interconnect-driven design, because stacking silicon layers allows more cells to be placed close to one another, thereby decreasing the average interconnect length [10]. A recent example of a two-tier system demonstrated a 15% reductions in both delay and power over the traditional 2DIC case [14]. One study suggested that 3D integration with 5 tiers can improve system timing by up to two technology generations [39]. Hua et.al. [40] demonstrate a thermally-aware design flow which analyzes temperature along with timing, area and power.

Hardware FFT architectures require large first-in-first-out (FIFO) memories, which can lead to large latencies. Much of the research in 3D design suggests that a big advantage of 3DIC technology is the ability to reduce memory latency.

Loi et.al. [55] show that for a memory intensive CPU-based system, 3DICs can be used to integrate the CPU core and memory on the same die, thus eliminating the need for external input/output pads and removing the limitation on the width of memory buses. They also demonstrate that the vertical on-chip buses in 3DIC Technology can support a high frequency ($\approx 2GHz$). This increased bus frequency and bus width leads to significant improvement in the system performance. A first-order performance study of cache memory with wafer-level 3D Integration by Zeng et.al. [75] has predicted that 3D technology can improve the latencies of cache memory by as much as 50%. This study had its emphasis on large caches in deep-submicron

technologies which gels in well with our goal of designing large FIFOs.

We will be using the MIT Lincoln Lab $0.18\mu m$ three-tier, three metal layer per tier, fully depleted silicon on insulator (FDSOI) process [67] to fabricate and analyze the system performance. The inter-tier via made available in this technology is roughly the size of a standard cell.

## 1.2 Goals of this work

We demonstrate an EDA flow to design circuits using multi-tier 3DIC technology. The goal of this work is to show how 3DIC technology can be used to improve the performance of designs using FIFO memories in pipeline FFT processors.

The EDA flow can be used for complete RTL-to-GDSII layer generation for 3DICs. Commercially proven single-tier tools are used for most of the steps in the flow which increases its robustness and gives a shorter path to transition from single-tier to multi-tier designs.

The 3DIC flow is demonstrated using two designs which have a large amount of memory. The first design is a 16KB capacity First-in-First-out (FIFO) buffer which uses only single-port memories and still offers stall-free operation as long as its capacity is not exhausted. It takes advantage of 3DIC technology to split the FIFO memory into more than one tier and improve latency of each sub-memory by virtue of reduced size.

The second design demonstrated is a 8192-point Fast Fourier Transform processor with a 12-bit real and 12-bit imaginary data bus. Again, the same circuit is syn-

thesized, placed and routed in the conventional way (single-tier) and the multi-tier manner to show design improvements. The designs are compared with other 8192-point FFT processors in technical literature. The improvement in performance is due to reduced sizes of memories, thus improving their access times and also making the design compact so that average wirelengths are reduced.

# Chapter 2

# FFT Algorithms and Architectures

This chapter introduces the Fast Fourier Transform (FFT) Algorithm and reviews the common hardware architectures that are used to compute the FFT algorithms.

## 2.1 Discrete Fourier Transform

The Discrete Fourier Transform is a procedure, or process, that can analyze the data points of a "digitized" time domain function to determine a series of sinusoids which, when summed together, reproduce the data points of the original function. The resulting Digital Fourier series is a valid expression of the original function [76].

Mathematically, the definition of the DFT is shown in Equation 2.1:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad 0 \leq k \leq N-1 \tag{2.1}$$

where the coefficient $W_N^{nk}$ (also called the twiddle factor) is a complex number

given by

$$W_N^{nk} = e^{-j2\pi\frac{nk}{N}} \tag{2.2}$$

In summary, the DFT is a matrix-vector product. If $(x_0, x_1, x_2, \ldots, x_{N-1})^T$ is the vector of input samples and $(X_0, X_1, X_2, \ldots, X_{N-1})^T$ is the vector of transformed values, the DFT can be written as

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 1 & W_N^1 & W_N^2 & W_N^3 & \ldots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & W_N^6 & \ldots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & W_N^{3(N-1)} & \ldots & W_N^{(N-1)(N-1)} \end{bmatrix} * \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} \tag{2.3}$$

Thus, in general, for an N-point DFT, we need to calculate N complex outputs from N complex inputs. The number of points and the format of the inputs and outputs (bit-width, fixed or floating point, etc.) is determined by the application.

The twiddle factors all have an amplitude of 1 and are also known as roots of unity. To illustrate the twiddle factors visually, we take an input sequence with 8 samples and represent the twiddle factor as a vector in the unit circle.

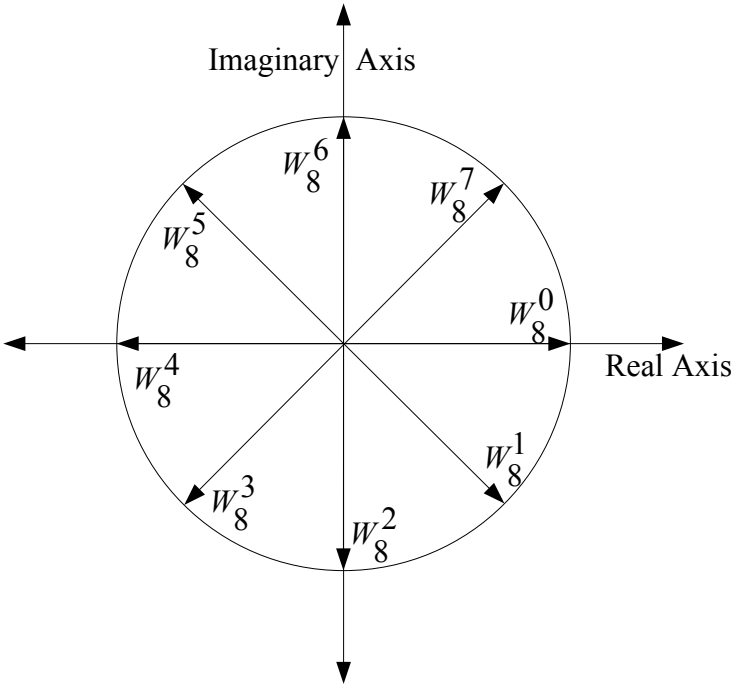


Figure 2.1: Twiddle Factors for 8-pt. DFT

$$
\begin{aligned}
\text{since} \quad e^{-j\theta} &= e^{-j(\theta+2m\pi)} \quad \text{where m is a whole number,} \\
\text{therefore} \quad e^{-j2\pi\frac{nk}{N}} &= e^{-j\left(\frac{2\pi nk}{N}+\frac{2m\pi N}{N}\right)} \\
&= e^{-j\frac{2\pi(nk+mN)}{N}} \\
&= e^{-j\frac{2\pi(nk \bmod N)}{N}} \\
\therefore \quad W_N^{nk} &= W_N^{(nk \bmod N)} \qquad (2.4)
\end{aligned}
$$

In general, the Equations 2.5-2.8 hold true for $W_N^{nk}$ where $nk$ is a whole number.

$$W_N^0 = -W_N^{N/2} = 1 \tag{2.5}$$

$$W_N^{N/4} = -W_N^{3N/4} = -j \tag{2.6}$$

$$W_N^{N/8} = -W_N^{5N/8} = \frac{\sqrt{2}}{2}(1-j) \tag{2.7}$$

$$W_N^{3N/8} = -W_N^{7N/8} = -\frac{\sqrt{2}}{2}(1+j) \tag{2.8}$$

These relationships between the twiddle factors, where some of them can be derived from others using trivial multiplications such as 1, -1, $j$ and $-j$, are used by different algorithms to speed up the computation time of the Discrete Fourier Transform as shown in section 2.2.

## 2.2 Fast Fourier Transform

Cooley and Tukey [22] first demonstrated a method to speedup the computation of a DFT by removing redundant computations. Thereafter, any algorithm which speeds up the computation of DFT has come to be categorized as the Fast Fourier Transform (FFT).

In general, the FFT can make the simplest and most effective optimizations when the number of samples to be transformed is an exact power of two, for which it can eliminate many unnecessary operations. The results of the FFT are the same as with the DFT; the only difference is that the algorithm usually runs much faster and with less resources. In algorithmic terms, the DFT takes $O(N^2)$ arithmetic operations to be computed, whereas the FFT takes $O(N \log N)$ arithmetic operations.

The most basic FFT algorithm is the Radix-2 Decimation in Frequency Algorithm. This algorithm decomposes even and odd-indexed frequency samples as shown mathematically in Equation Sets 2.9 and 2.10:

$$\begin{aligned}
X(2k) &= \sum_{n=0}^{N-1}\left[x(n)W_N^{2kn}\right] \\
&= \sum_{n=0}^{\frac{N}{2}-1}\left[x(n)W_N^{2kn}\right] + \sum_{n=0}^{\frac{N}{2}-1}\left[x\left(n+\frac{n}{2}\right)W_N^{2k\left(n+\frac{N}{2}\right)}\right] \\
&= \sum_{n=0}^{\frac{N}{2}-1}\left[x(n)W_N^{2kn}\right] + \sum_{n=0}^{\frac{N}{2}-1}\left[x\left(n+\frac{n}{2}\right)W_N^{2kn}1\right] \\
&= \sum_{n=0}^{\frac{N}{2}-1}\left[\left(x\left(n\right)+x\left(n+\frac{N}{2}\right)\right)W_{\frac{N}{2}}^{kn}\right] \\
&= DFT_{\frac{N}{2}}\left[x\left(n\right)+x\left(n+\frac{N}{2}\right)\right] \qquad (2.9)
\end{aligned}$$

$$\begin{aligned}
X(2k+1) &= \sum_{n=0}^{N-1}\left[x(n)W_N^{(2k+1)n}\right] \\
&= \sum_{n=0}^{\frac{N}{2}-1}\left[\left(x\left(n\right)+W_N^{N/2}x\left(n+\frac{N}{2}\right)\right)W_N^{(k+1)n}\right] \\
&= \sum_{n=0}^{\frac{N}{2}-1}\left[\left(x\left(n\right)-x\left(n+\frac{N}{2}\right)\right)W_N^{n}W_{\frac{N}{2}}^{kn}\right] \\
&= DFT_{\frac{N}{2}}\left[\left(x\left(n\right)-x\left(n+\frac{N}{2}\right)\right)W_N^{n}\right] \qquad (2.10)
\end{aligned}$$

For an 8-pt FFT, this can be pictorially demonstrated as shown in Fig.2.2.

As can be seen, the set of equations given by $\left[x(n)+x\left(n+\frac{N}{2}\right)\right]$ and $\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^n$ appear frequently during the computations. These can be calculated using a standard circuit known as the butterfly. As this butterfly processes 2 samples at a time, the architecture uses what is known as the Radix-2 FFT

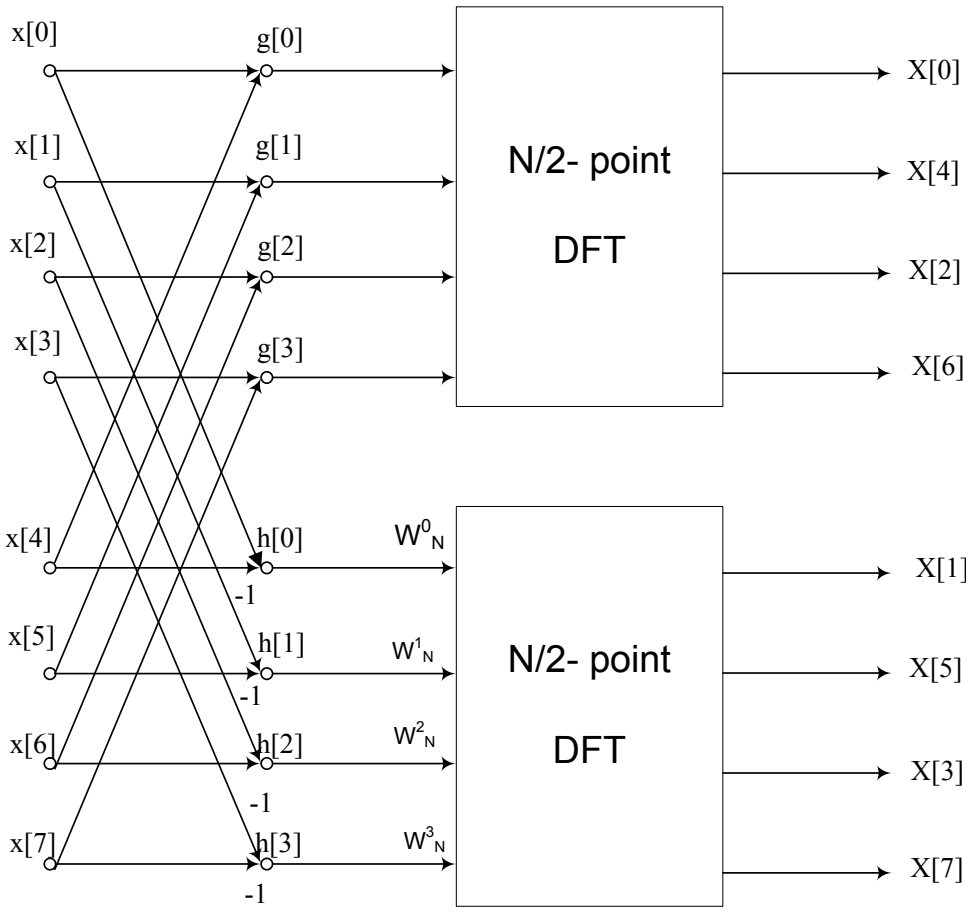


Figure 2.2: 8-pt Radix-2 FFT using Decimation in Frequency Algorithm [43]

algorithm. For a Radix-2 DIF FFT Algorithm, the butterfly is as shown in Fig.2.3.

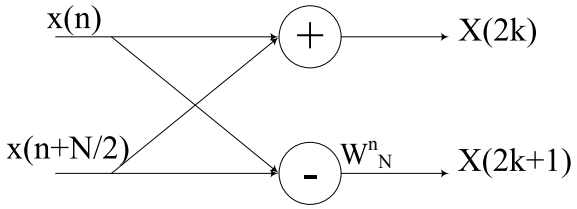


Figure 2.3: Butterfly of DIF Radix-2 FFT

Recursively applying Equation Sets 2.9 and 2.10 to all the samples for an 8-pt DFT gives the signal flow graph as shown in Fig 2.4.

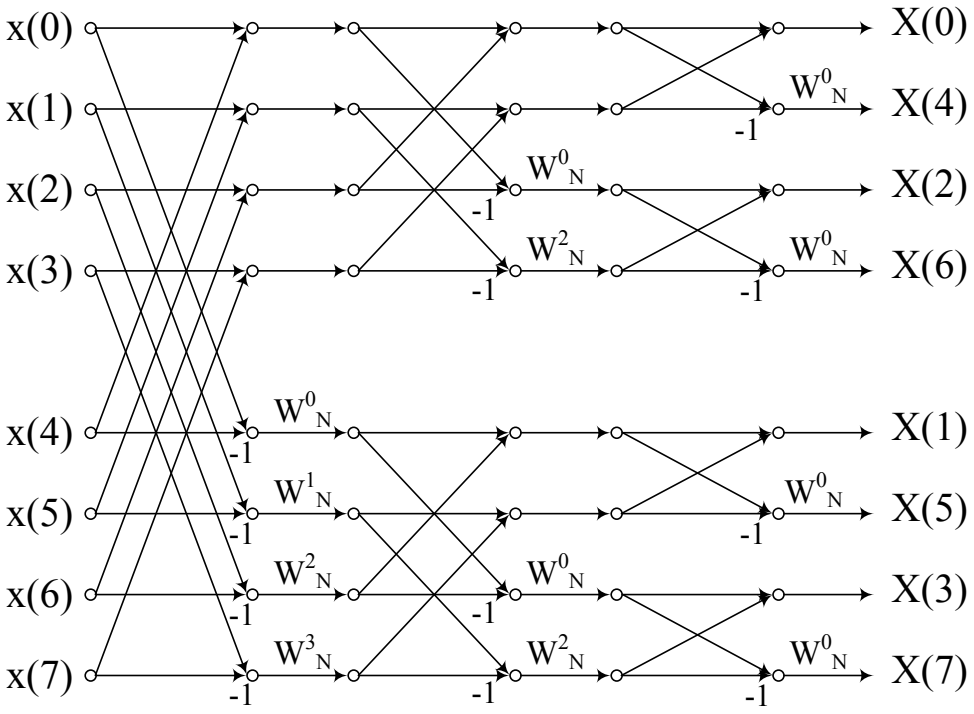


Figure 2.4: 8-pt Radix-2 FFT using Decimation in Frequency Algorithm

The original DFT equation can be similarly rearranged using different radices, thus giving rise to various FFT algorithms as described in Chapter 2. Each of these algorithms and the architectures used to execute them could serve varying interests according to the intended application. The focus of this work is high-speed large-point FFTs and we choose the Radix 2/4/8 Algorithm and the multi-path delay commutator Pipelined Architecture for our 8192-pt FFT design.

## 2.3 FFT Algorithms

This section describes in brief some of the popular variations [20] of the basic Cooley-Tukey Radix-2 FFT algorithm described in the previous chapter.

### 2.3.1 Radix-4 Algorithm

As shown in Section 2.2, decomposing the frequency samples into 2 bins (one each for odd and even samples) gives the Radix-2 Algorithm. Increasing the number of bins to 4 gives the Radix-4 Algorithm [60]. It is mathematically expressed [54] as shown in Eq.set 2.11.

$$\begin{aligned} X(4k_1+l) &= \sum_{k=0}^{\frac{N}{4}-1}\left[x(n)+x\left(n+\frac{N}{4}\right)W_4^l+x\left(n+\frac{N}{2}\right)W_4^{2l}+x\left(n+\frac{3N}{4}\right)W_4^{3l}\right] \\ &\quad *W_N^{nl}W_{\frac{N}{4}}^{nk_1} \\ l &= 0,1,2,3; \quad k_1 = 0,1,2,3\ldots\frac{N}{4}-1 \end{aligned} \tag{2.11}$$

The Butterfly for the Radix-4 Algorithm [20] is shown in Fig.2.5.

While radix-2 and radix-4 FFTs are the most widely used FFT algorithms, it is also possible to design FFTs with even higher radix butterflies. The reason they are not often used is because the control and dataflow of their butterflies are more complicated and the additional efficiency gained diminishes rapidly for radices greater than four [8]. The radix-8 algorithm is described here for completeness. The other algorithms try to achieve the benefits of higher-radix FFTs (lower multiplications required) even though they use structures which have lesser complexity similar to radix-2 or radix-4 algorithms.

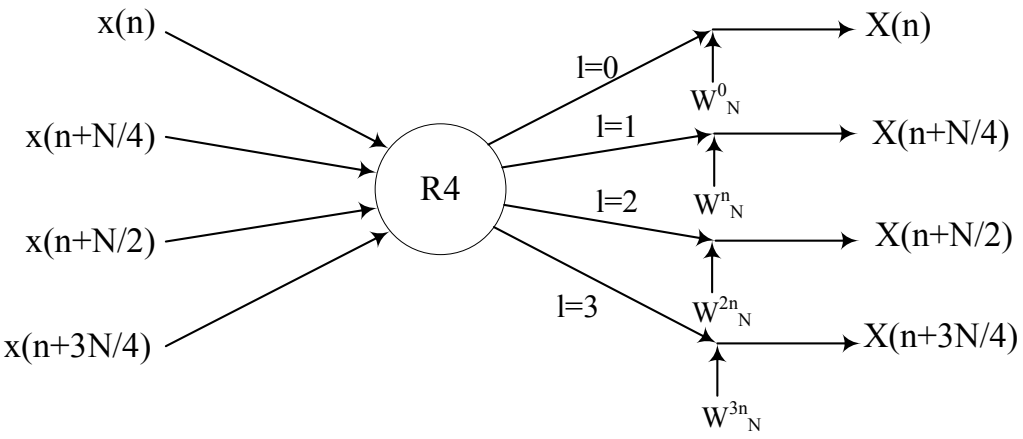


Figure 2.5: Butterfly for Radix-4 Algorithm

### 2.3.2 Radix-$2^2$ Algorithm

He and Torkelson [34] demonstrated the Radix-$2^2$ Algorithm by rearranging 4 samples together in a slightly different manner than the Radix-4 Algorithm. The Radix-$2^2$ algorithm has the same multiplicative complexity of Radix-4 algorithms but has a signal flow graph similar to the Radix-2 algorithm.

It is mathematically expressed [54] as shown in Eq.set 2.12

$$
\begin{aligned}
&X(4k_1 + 2l_2 + l_1) \\
&\quad = \sum_{k=0}^{\frac{N}{4}-1} \Bigg[ x(n) + x\left(n + \frac{N}{4}\right) W_4^{2l_2+l_1} + x\left(n + \frac{N}{2}\right) W_4^{4l_2+2l_1} \\
&\qquad\qquad + x\left(n + \frac{3N}{4}\right) W_4^{6l_2+3l_1} \Bigg] \quad * \quad W_N^{n(2l_2+l_1)} W_{\frac{N}{4}}^{nk_1} \\
&\quad = \sum_{k=0}^{\frac{N}{4}-1} \Bigg\{ \left[ x(n) + (-1)^{l_1} x\left(n + \frac{N}{2}\right) \right] \\
&\qquad + (-1)^{l_2} (-j)^{l_1} \left[ x\left(n + \frac{N}{4}\right) + (-1)^{l_1} x\left(n + \frac{3N}{4}\right) \right] \Bigg\} \quad * \quad W_N^{n(2l_2+l_1)} W_{\frac{N}{4}}^{nk_1}
\end{aligned}
$$

$$
l_1, l_2 \;=\; 0, 1; \quad k_1 = 0, 1, 2, 3 \ldots (N/4) - 1 \tag{2.12}
$$

The Radix-$2^2$ Butterfly is as shown in Fig 2.6

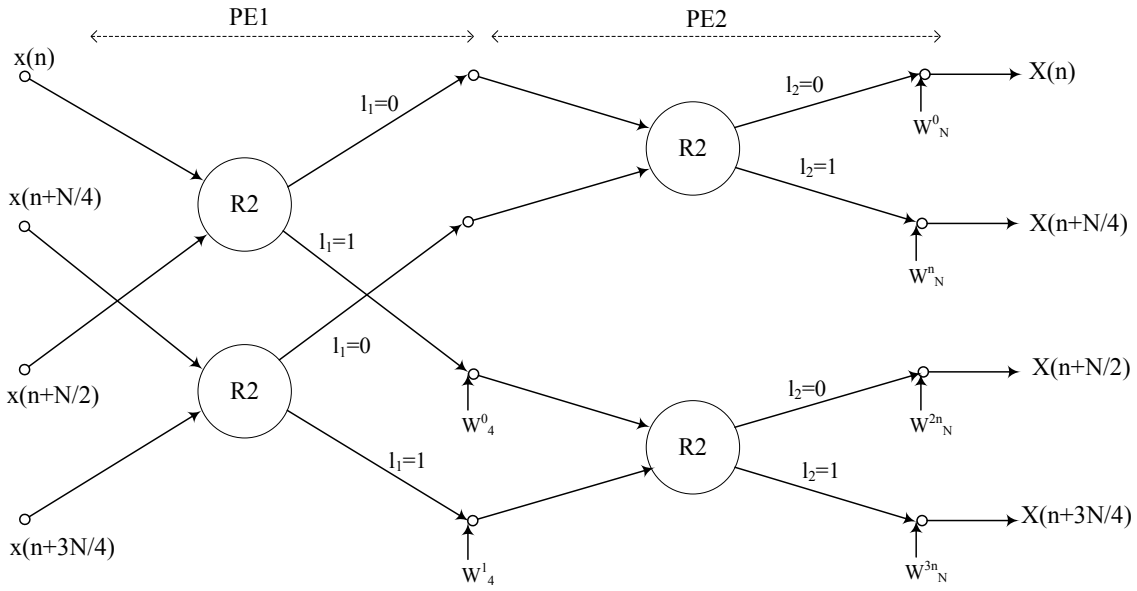


Figure 2.6: Butterfly for Radix $2^2$ Algorithm

### 2.3.3 Radix-8 Algorithm

Grouping 8 samples together gives the Radix-8 Algorithm [11]. Mathematically, it is expressed as shown in Eq. set 2.13.

$$
\begin{aligned}
& X(8k+l) \\
&= \sum_{n=0}^{N-1} x(n) W_N^{(8k+l)n} \\
&= \sum_{m=0}^{\frac{N}{8}-1} x\left(n+\frac{mN}{8}\right) W_N^{(8k+l)\left(n+\frac{mN}{8}\right)} \\
&= \sum_{m=0}^{7} \sum_{n=0}^{\frac{N}{8}-1} \left[x\left(n+\frac{mN}{8}\right) W_8^{lm}\right] W_N^{nl} W_{\frac{N}{8}}^{nk} \\
&= \sum_{n=0}^{\frac{N}{8}-1} \left\{ \left[x(n) + x\left(n+\frac{2N}{8}\right) W_4^l + x\left(n+\frac{4N}{8}\right) W_4^{2l} + x\left(n+\frac{6N}{8}\right) W_4^{-l}\right] \right. \\
&\left. + \left[x\left(n+\frac{N}{8}\right) + x\left(n+\frac{3N}{8}\right) W_4^l + x\left(n+\frac{5N}{8}\right) W_4^{2l} + x\left(n+\frac{7N}{8}\right) W_4^{-l}\right] W_8^l \right\} \\
&\quad * W_N^{nl} W_{\frac{N}{8}}^{nk}
\end{aligned}
$$

$$
l = 0,1,2,3,4,5,6,7; \quad k = 0,1,2,3 \ldots (N/8) - 1 \tag{2.13}
$$

The Radix-8 Butterfly is as shown in Fig 2.7

### 2.3.4 Radix-2/4 Algorithm

Split Radix algorithms [27] in general use a separate decomposition for the even and odd samples. For example, the split radix 2/4 Algorithm uses a radix-2 decom-

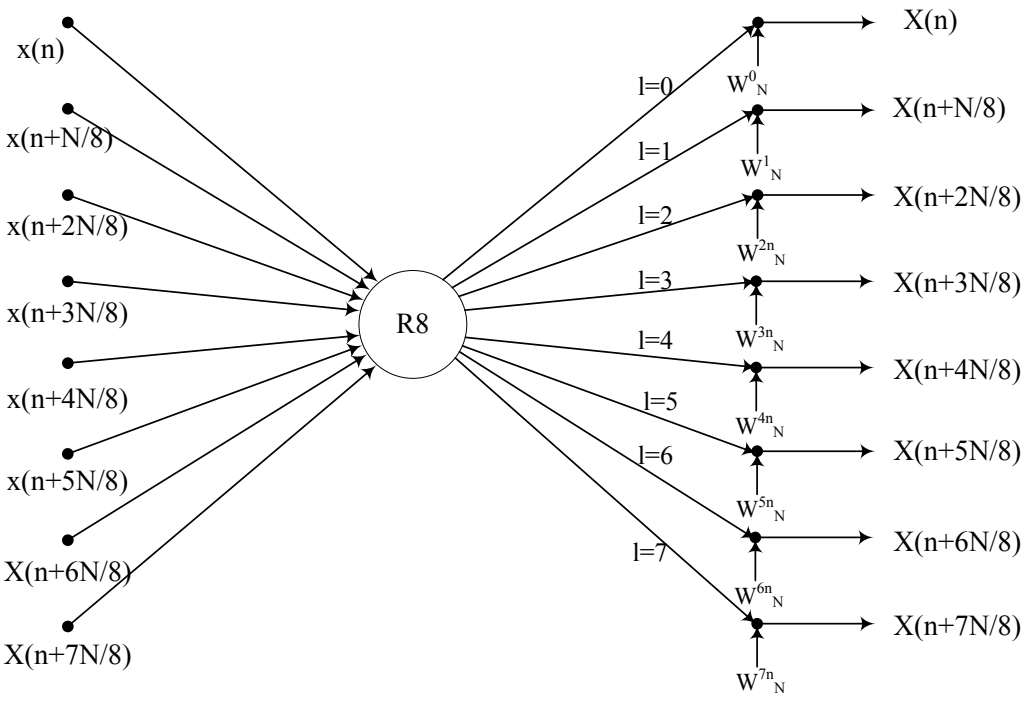


Figure 2.7: Butterfly for Radix 8 Algorithm

position for the even samples and a radix-4 decomposition for the odd samples. The advantage of this algorithm is that it provides the lowest known number of operations for computing length-$2^n$ FFTs [28]. The disadvantage is that the resulting structure

is irregular. This algorithm is mathematically expressed as shown in Eq. set 2.14.

$$
\begin{aligned}
X(2k) &= \sum_{n=0}^{\frac{N}{2}-1} \left[ x(n) + x\left(n + \frac{2N}{4}\right) \right] W_{\frac{N}{2}}^{nk} \\
X(4k_1+1) &= \sum_{n=0}^{\frac{N}{4}-1} \left\{ x(n) - x\left(n + \frac{2N}{4}\right) - j\left[ x\left(n + \frac{N}{4}\right) - x\left(n + \frac{3N}{4}\right) \right] \right\} \\
&\quad * W_N^n W_N^{4nk_1} \\
X(4k_1+3) &= \sum_{n=0}^{\frac{N}{4}-1} \left\{ x(n) - x\left(n + \frac{2N}{4}\right) + j\left[ x\left(n + \frac{N}{4}\right) - x\left(n + \frac{3N}{4}\right) \right] \right\} \\
&\quad * W_N^{3n} W_N^{4nk_1}
\end{aligned}
$$

$$
k = 0, 1, 2, 3 \ldots (N/2) - 1; \quad k_1 = 0, 1, 2, 3 \ldots (N/4) - 1 \tag{2.14}
$$

The Radix-2/4 Butterfly is as shown in Fig 2.8

### 2.3.5 Radix-2/8 Algorithm

Using the Radix-8 decomposition for the odd samples instead of Radix-4 decomposition gives the Split-Radix 2/8 Algorithm. Mathematically, it is expressed as shown

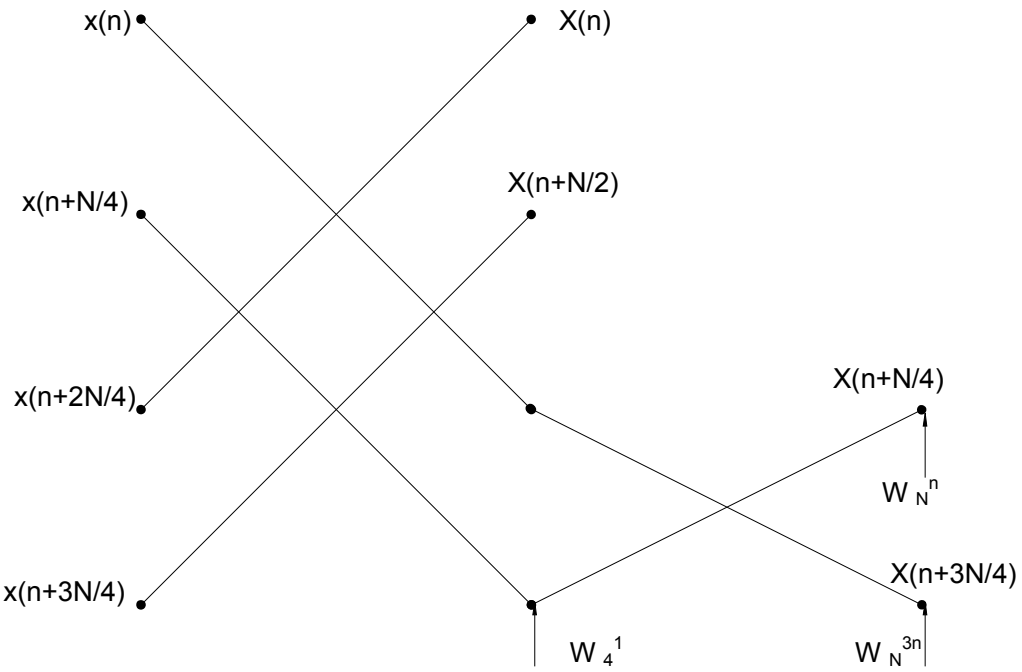


Figure 2.8: Butterfly for Radix-2/4 Algorithm

in Eq.2.15.

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} \left[ x(n) + x\left(n + \frac{2N}{4}\right) \right] W_{\frac{N}{2}}^{nk}$$

$$X(8k_1 + l) =$$

$$\sum_{n=0}^{\frac{N}{8}-1} \left\{ \left[ x(n) + x\left(n + \frac{2N}{8}\right) W_4^l + x\left(n + \frac{4N}{8}\right) W_4^{2l} + x\left(n + \frac{6N}{8}\right) W_4^{-l} \right] + \left[ x\left(n + \frac{N}{8}\right) + x\left(n + \frac{3N}{8}\right) W_4^l + x\left(n + \frac{5N}{8}\right) W_4^{2l} + x\left(n + \frac{7N}{8}\right) W_4^{-l} \right] W_8^l \right\} * W_N^{nl} W_{N/8}^{nk_1}$$

$$l = 1, 3, 5, 7; \quad k = 0, 1, 2, 3 \ldots (N/2) - 1; \quad k_1 = 0, 1, 2, 3 \ldots (N/8) - 1 \tag{2.15}$$

The Radix-2/8 Butterfly is as shown in Fig 2.9

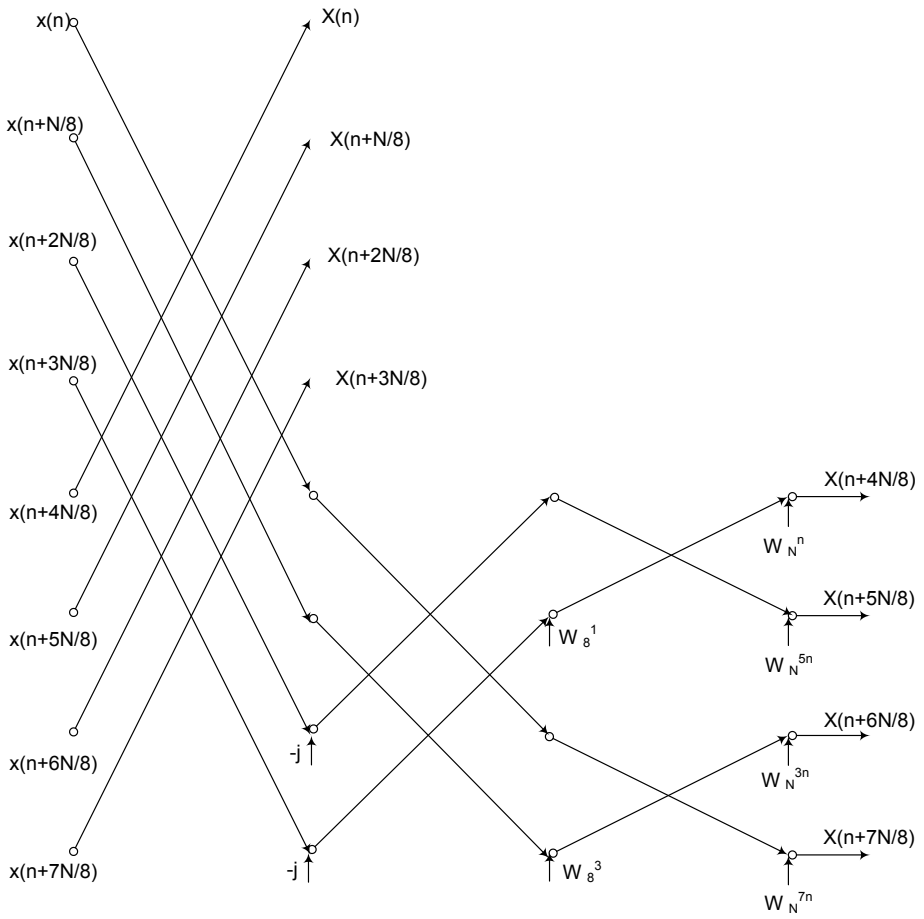


Figure 2.9: Butterfly for Radix-2/8 Algorithm

### 2.3.6 Radix-2/4/8 Algorithm

Rearranging 8 samples together in a different manner gives the Radix-2/4/8 Algorithm [41]. Mathematically, it is expressed as:

$$
\begin{aligned}
&X(8k + 4l_3 + 2l_2 + l_1) \\
&= \sum_{n=0}^{\frac{N}{8}-1} \Bigg\{ \left[ x(n) + x\left(n + \frac{2N}{8}\right) W_4^l + x\left(n + \frac{4N}{8}\right) W_4^{2l} + x\left(n + \frac{6N}{8}\right) W_4^{-l} \right] \\
&+ \left[ x\left(n + \frac{N}{8}\right) + x\left(n + \frac{3N}{8}\right) W_4^l + x\left(n + \frac{5N}{8}\right) W_4^{2l} + x\left(n + \frac{7N}{8}\right) W_4^{-l} \right] W_8^l \Bigg\} \\
&\qquad * W_N^{nl} W_{\frac{N}{8}}^{nk} \\
&= \sum_{n=0}^{\frac{N}{8}-1} \Bigg\{ \left[ \left( x(n) + W_2^{l_1} x\left(n + \frac{4N}{8}\right) \right) + W_2^{l_2} W_4^{l_1} \left( x\left(n + \frac{2N}{8}\right) + W_2^{l_1} x\left(n + \frac{6N}{8}\right) \right) \right] \\
&+ \left[ \left( x\left(n + \frac{N}{8}\right) + W_2^{l_1} x\left(n + \frac{5N}{8}\right) \right) + W_2^{l_2} W_4^{l_1} \left( x\left(n + \frac{3N}{8}\right) + W_2^{l_1} x\left(n + \frac{7N}{8}\right) \right) \right] \\
&\qquad * W_8^{2l_2 + l_1} \Bigg\} * W_N^{n(4l_3 + 2l_2 + l_1)} W_{\frac{N}{8}}^{nk}
\end{aligned}
$$

$$l_1, l_2, l_3 = 0, 1; \quad k = 0, 1, 2, 3 \ldots (N/8) - 1$$

The signal flow graph for the Radix 2/4/8 algorithm is as shown in Fig 2.10.

Apart from these popular algorithms, researchers have used many other algorithms to build FFT hardware. These include the prime factor algorithm [17], Winograd algorithm [71], mixed-radix FFT [63], algorithms with radix greater than 8 [12, 2] and the Radix$2^4$ [74] Algorithm. Specialized algorithms for real-valued data have

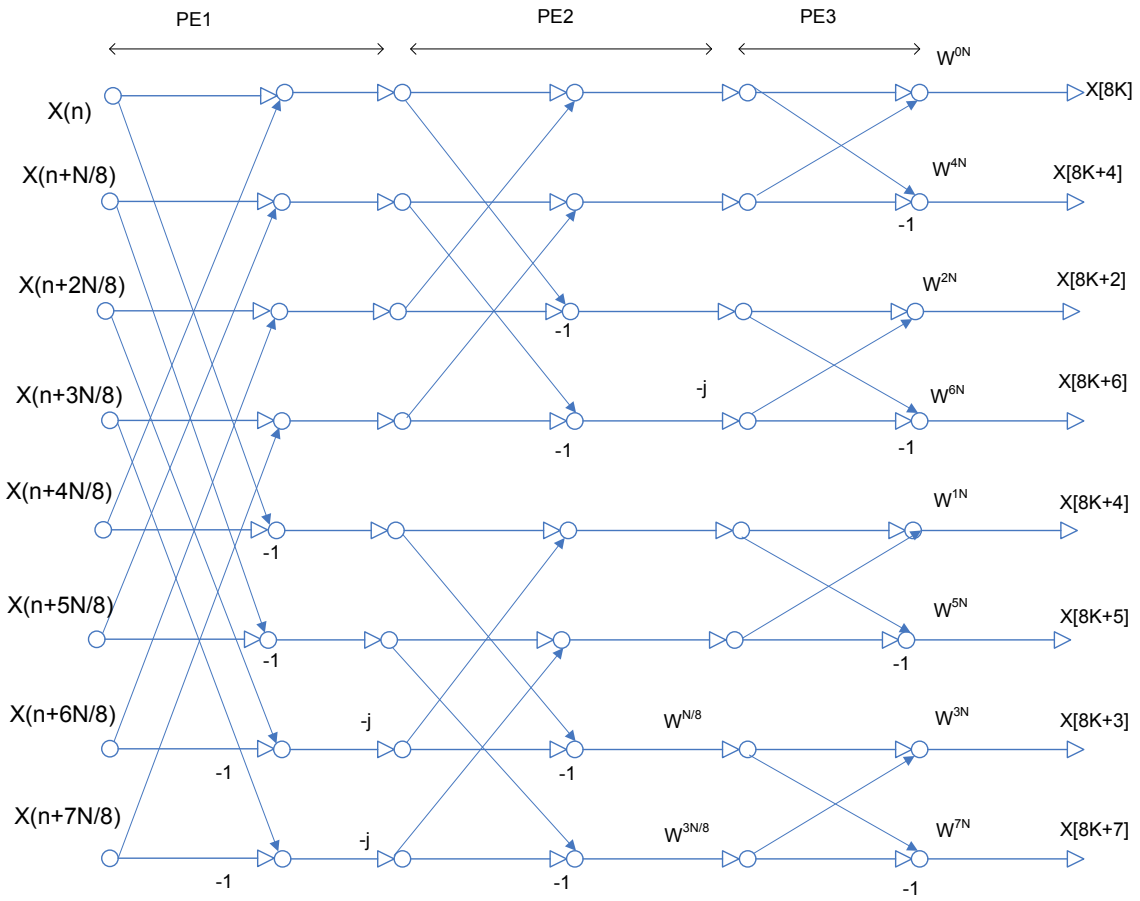


Figure 2.10: Signal Flow Graph for Radix 2/4/8 Algorithm

also been developed [65].

Our proposed FFT Architecture is based on the Radix-2/4/8 Algorithm and closely resembles the low-power variable-length fast Fourier transform processor by Lin et.al. [52].

Section 2.4 will compare different FFT Architectures in terms of speed, complexity and memory and area requirements.

## 2.4 FFT Architectures

This section will review common FFT structures used in literature [57].

### 2.4.1 Sequential Processor

The basic sequential processor consists of a processing element (PE) that can compute a butterfly. The same memory can be used to store the data, intermediate results and the twiddle factors. The amount of hardware involved is very small and it takes $(N/2)\log_2 N$ sequential operations to compute the FFT.

### 2.4.2 Pipeline Processor

To improve the performance of the sequential processor, parallelism can be introduced by using a separate arithmetic unit for each stage of the FFT. This increases the throughput by a factor of $\log_2 N$ when the different units are pipelined. This architecture is also known as cascaded FFT architecture and will be used in our proposed design. Early examples of this approach are [32] and [59].

### 2.4.3 Parallel Iterative Processor

By adding more processing elements to the processor in each sequential pipeline stage, performance can be improved even further. The butterflies can then be computed in parallel in any stage. The total execution time for the parallel iterative processor is $\log_2 N$ cycles. Bungard et.al. [15] used this scheme in their FFT proces-

sor for radar signal processing.

### 2.4.4 Array Analyzer

A fully parallel structure can be constructed by having a PE for each of the butterfly operations. This involves a lot of hardware and is not an attractive option for a large N. Examples of such structures for small-point FFTs are built in [19] and [50].

## 2.5 Pipeline FFT structures

Pipeline FFTs are a class of parallel algorithms that contain an amount of parallelism equal to $\log_R N$ where $N$ is the number of points for an FFT and $R$ is the radix. A Pipeline implementation of the FFT was first proposed in [32] which consisted of a series of computational blocks each composed of delay lines, coefficient storage, commutators, multipliers, and adders. Pipeline FFTs can be generally run at high-speeds and the amount of pipelining increased or decreased to meet timing.

He and Torkelson [34] have classified different pipeline approaches and put into functional blocks with unified terminology.

### 2.5.1 Multi-path delay commutator

Radix-2 Multi-path Delay Commutator is the most classical approach for pipeline implementation of radix-2 FFT algorithm. As shown in Fig. 2.11, the input sequence

has been broken into two parallel data stream flowing forward, with correct *distance* between the data elements entering the butterfly scheduled by proper delays. Both butterflies and multipliers are in 50% utilization. In certain applications like Multiple-input Multiple-output (MIMO) Antenna systems, the first commutator at the input data stream can be can be replaced with 2 such commutators to keep the pipeline components 100% utilized. The output data rate for this structure is twice the clock frequency and that is what makes it appealing for high-speed applications.

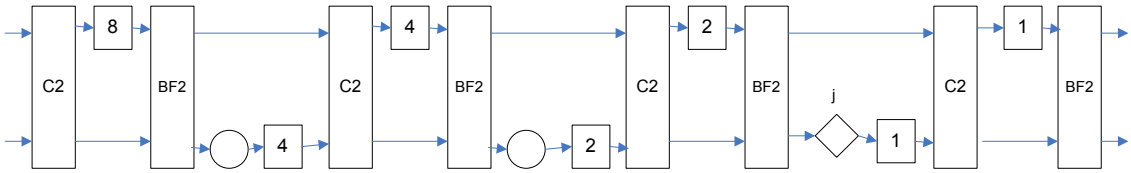


Figure 2.11: R2MDC (N=16)

In Fig. 2.11, the block labeled C2 is a 2-input, 2-output crisscross commutator structure which routes each of the inputs in one of the unique outputs. In the diagrams to follow, blocks labeled C$X$ stand for an $X$-input, $X$-output Crisscross commutator. Each of the inputs is routed to one of the unique outputs. The second operand for each of the multipliers is a twiddle factor which is not shown for clarity.

The Radix-4 Multi-path Delay Commutator as shown in Fig.2.12 is a radix-4 version of R2MDC. However, it suffers from low, 25%, utilization of all components. Again, this can be compensated in applications like MIMO where 4 FFTs can be processed simultaneously. It requires $3 \log N$ multipliers, $logN$ full radix-4 butterflies and $2.5N - 4$ memory elements.

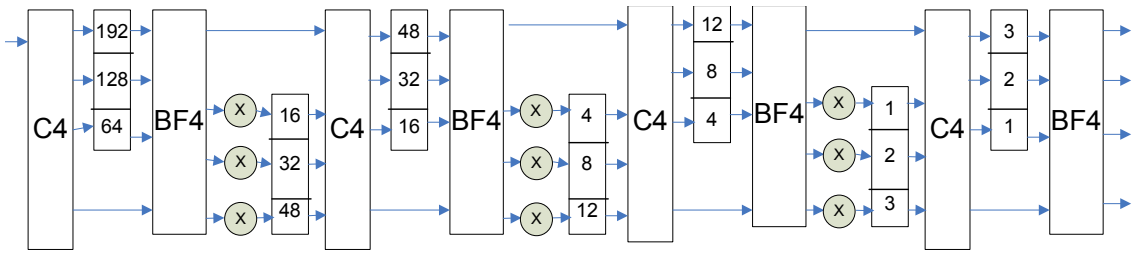


Figure 2.12: R4MDC (N=256)

### 2.5.2 Single-path delay feedback

Radix-2 Single-path Delay Feedback (Fig.2.13) uses the registers more efficiently by storing one output of each butterfly in feedback shift registers. A single data stream goes through the multiplier at every stage. It has same number of butterfly units and multipliers as in R2MDC approach, but with reduced memory requirement: N - 1 registers. Its data throughput is (1/x) times that of the corresponding RxMDC architecture. The lower area requirement for this structure compared to MDC structures is to its advantage.

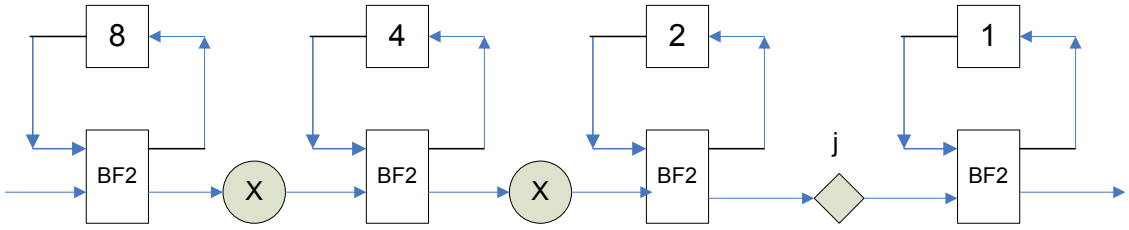


Figure 2.13: R2SDF (N=16)

Radix-4 Single-path Delay Feedback (Fig.2.14) was proposed as a radix-4 version of R2SDF, employing CORDIC iterations. The utilization of multipliers has been

increased to 75% by storing 3 out of 4 butterfly outputs. However, the utilization of the radix-4 butterfly, which is fairly complicated and contains at least 8 complex adders, is dropped to only 25%. It requires $\log N - 1$ multipliers, log, N full radix-4 butterflies and memory of size $N - 1$ words.

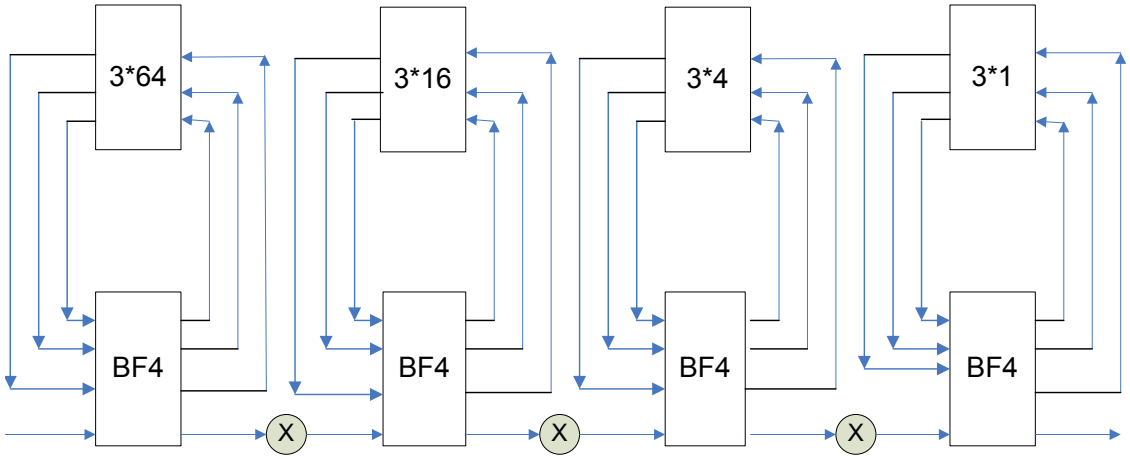


Figure 2.14: R4SDF (N=256)

### 2.5.3 Single-path delay commutator

Radix-4 Single-path Delay Commutator (Fig.2.15) uses a modified radix-4 algorithm with programmable 1/4 radix- 4 butterflies to achieve higher, 75% utilization of multipliers. Conceptually, it divides the Radix-4 butterfly into 4 parts and uses only one part of hardware and runs it at 4 times the original speed to achieve the same effect as a Radix-4 Butterfly. A multiplexed Delay-Commutator also reduces the memory requirement to $2N - 2$ from $2.5N - 1$, that of R4MDC. The butterfly and delay-commutator become relatively complicated due to programmability require-

ment.

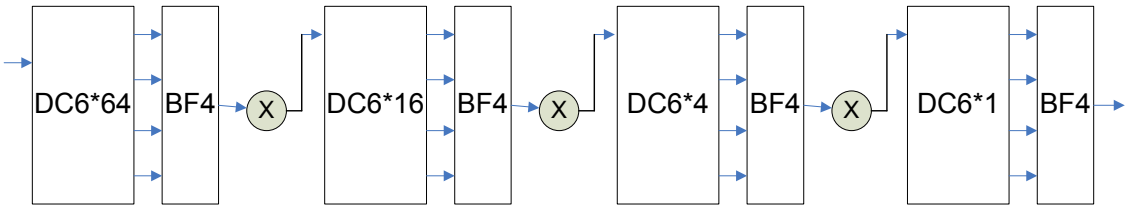


Figure 2.15: R4SDC (N=256)

The hardware requirements of these pipelined architectures are summarized in Table 2.1.

Table 2.1: Hardware Requirements of different FFT Architectures [35, 73, 74]

| Architecture | Complex Multiplies | Complex Additions | Memory Size | Butterfly |
|---|---|---|---|---|
| R2MDC | $2\,(\log_4 N - 1)$ | $4\log_4 N$ | $1.5N - 2$ | Radix-2 |
| R4MDC | $3\,(\log_4 N - 1)$ | $8\log_4 N$ | $2.5N - 4$ | Radix-4 |
| R4SDC | $(\log_4 N - 1)$ | $3\log_4 N$ | $2N - 2$ | Radix-4 |
| SRMDC | $4\,(\log_4 N - 1)$ | $12\log_4 N - 8$ | $1.5N - 2$ | Split-Radix (2/4) |
| R2SDF | $2\,(\log_4 N - 1)$ | $4\log_4 N$ | $N - 1$ | Radix-2 |
| R4SDF | $(\log_4 N - 1)$ | $8\log_4 N$ | $N - 1$ | Radix-4 |
| R$2^2$SDF | $(\log_4 N - 1)$ | $4\log_4 N$ | $N - 1$ | Radix-$2^2$ |
| R$2^3$SDF | $(\log_8 N - 1)$ | $2\log_2 N$ | $N - 1$ | Radix-$2^3$ |
| R$2^3$MDC | $2\,(\log_8 N - 1)$ | $2\log_2 N$ | $1.5N - 2$ | Radix-$2^3$ |

## 2.6 FFT Metrics

FFT Performance is measured using a number of metrics. While comparing FFT Processors, it is essential to consider their design technology node and the number of

points they transform. To make a fair comparison, only FFT processors which generate 8192-point FFT results are compared. We use the constant electric field scaling method [61] to adjust for technology process differences. In the next paragraph, we assume that the ratio of technology of the design to be compared to the technology we use i.e. $0.18\mu m$ is $K$. Thus,

$$K = \frac{Technology}{0.18\mu m} \tag{2.16}$$

These are some of the important metrics [8, 52] for measuring and comparing FFT processor performance:

1. *Execution Time*: The amount of time it takes for the FFT Processor to finish execution of one set of FFT (or one Discrete Fourier Transform). This is the metric which will decide the fastest FFT Processor in terms of a complete transform processed. Normalized execution time will scale by the same factor $(1/K)$ as the gate delay scales between the technologies.

2. *Latency*: The time difference between the time the first input samples are fed to the processor and the first output samples are present at the output. Processors with deeper pipelines will have a longer latency. If the design is such that it can work at its full throughput once the pipeline is filled, the latency might become inconsequential for comparison.

3. *Clock Frequency*: This is the maximum frequency at which the FFT Processor can be operated at without causing timing violations. A higher clock fre-

quency doesn't necessarily mean a better design. Architectures can improve their throughput by cranking up the clock if possible. Different architectures may need to run at different clock frequencies to give the same throughput. Clock Frequency will scale by $K$ between technologies.

4. *Power*: Power dissipated per unit of time is an important metric as it can indicate how much the temperature of the chip fabricated might increase. Especially for 3D designs where heat dissipation is a bigger problem compared to that for 2D designs, this is an important metric to consider. This also determines the battery life for mobile devices. Power scales by $1/K^2$ between technologies.

5. *Energy*: The amount of power dissipated per processing of a complete set of FFT Transform is an important metric for systems which have low energy requirements. This is the energy required to process one set of FFT. Since energy is the product of power and time, it will scale by product of their scaling factors which are $1/K^2$ and $1/K$ respectively. Thus, energy will scale by $1/K^3$ between technologies.

6. *Area*: The total die area occupied by the design. This will include the arithmetic elements and the internal memory needed for the execution of the transform. Since the length and breadth of the entire design will scale by $1/K$, the area will scale by $1/K^2$ between technologies.

7. *FFTs per energy*: This metric gives the energy efficiency of the processor by indicating the number of FFT transform sets that the processor can execute

per unit energy. This is given as

$$FFTs\ per\ energy = \frac{K^3}{Power\ of\ N\text{-}pt\ FFT\ *\ Execution\ Time} \tag{2.17}$$

This parameter normalizes the technology node out from the energy figure for each design compared. Although Y.W.Lin et.al [53] and Y.T.Lin et.al [52] scale this parameter by $1/K$, we feel it is more appropriate to use $1/K^3$ since all the technologies we compare are less than $0.6\mu m$ and we use the constant electric field scaling method [61]. $1/K$ represents fixed voltage scaling which is appropriate for scaling above the $0.6\mu m$ process technology.

8. *Energy * Time*: The drawback of considering energy consumption alone is that it gives no information about the speed of a processor. In order to more fully compare two designs in a single measure, the product $energy * time$ is often used, where *time* is the time required to perform a particular operation. Similar to some of the other metrics, we will normalize the technology node from the equation:

$$Energy\ *\ Time = \frac{Execution\ Time\ for\ N\text{-}pt\ FFT}{FFTs\ per\ energy} \tag{2.18}$$

A more general approach considers merit functions of the form $energy^x * time^y$ and $energy^x * time^y * area^z$ where $x$, $y$, and $z$ vary depending on the relative importance of the parameters in a particular application[8].

## 2.7 Related work

To demonstrate the 3DIC technology, we decided to design an FFT processor with a large number of points and that which could be easily manageable with our current EDA tools. Also, it needed to have some reference designs in technical literature to compare with.

Since OFDM-based applications require FFT processors with varying number of points from 512 points to 8192 points, many designs focused on this range. A wide variety of different pipeline architectures and algorithms have been used for FFT designs processing this range of points. For example, 1024-point FFT processors have been designed using the R4SDC [51], R2$^2$SDF [36] and R2SDF [72] architectures. Y.T.Lin et al. [52] use the Radix-2/4/8 architecture to process variable lengths of FFTs (2048, 1024, 512 points) using the same hardware.

We chose the 8192-point FFT as it was felt to be big enough to be an interesting design and three reference designs were present in technical literature which it could be compared with. Table 2.2 shows the performance characteristics of 8192-point FFT processors by Jia [41], Bidet [13] and Lin [53]. The vertical arrows shown in front of some of the properties indicate which is the preferred direction for the particular property. For example, a down arrow in front of the "Power" field indicates that a lower power number is better.

The single path delay feedback or single path delay commutator architectures are generally the preferred choice for designing pipeline FFT processors. The multi-path architectures usually require much more memory than the corresponding single-path

Table 2.2: Previous 8192-pt FFT Processors

| | Jia [41] | Bidet [13] | Lin [53] |
|---|---|---|---|
| Word length (bits) | 12 | 12 | 11 |
| Radix | 2/4/8 | 4 | 2/4/8 |
| Process, $\mu m$ | 0.6 | 0.5 | 0.18 |
| $K$ | 3.33 | 2.77 | 1 |
| Voltage, V | 3.3 | 3.3 | 1.8 |
| Power, mW $\downarrow$ | 650 | 600 | 25.2 |
| Norm. Power, mW $\downarrow$ | 58.5 | 77.7 | 25.2 |
| Clock Freq, MHz $\uparrow$ | 20 | 20 | 20 |
| Norm. Clock Freq, MHz $\uparrow$ | 66.6 | 55.5 | 20 |
| Execution Time, $\mu s$ $\downarrow$ | 400 | 400 | 717 |
| Norm. Exec Time $\mu s$ $\downarrow$ | 120 | 144 | 717 |
| Energy, $\mu J$ $\downarrow$ | 260 | 240 | 18.07 |
| Norm. Energy $\mu J$ $\downarrow$ | 7 | 112 | 18.07 |
| Area, $mm^2$ $\downarrow$ | 107 | 100 | 4.84 |
| Norm. Area, $mm^2$ $\downarrow$ | 9.63 | 12.96 | 4.84 |
| FFT/energy, $mJ^{-1}$ $\uparrow$ | 12.54 | 11.57 | 55.32 |
| Norm. F/E, $mJ^{-1}$ $\uparrow$ | 139.3 | 89.2 | 55.32 |
| Energy * Time (nJ*ns) $\downarrow$ | 104000 | 96000 | 12955 |
| Norm. E * T (nJ*ns) $\downarrow$ | 842.4 | 1612.4 | 12955 |

architectures. But the multi-path architecture has better throughput compared to the single-path ones and hence we prefer the MDC architecture for our proposed design. The 2/4/8 Radix is chosen as the Butterfly of our design. The reason for these choices is better explained in chapter 5. To our knowledge, the proposed design is the first one using the Radix 2/4/8 Butterfly with the MDC Architecture.

# Chapter 3

# 3DIC Flow

This chapter describes the Electronic Design Automation(EDA) flow that we used for the synthesis and physical design of the FIFO and FFT as described in later chapters. Section 3.1 introduces the flow and compares it with other approaches. Section 3.2 describes the standard cell library designed in our target 3DIC technology. Section 3.3 shows the methodology to generate dummy memory models for our experiments. Section 3.4 explains the various steps in the flow.

## 3.1 Introduction to the 3DIC flow

The EDA flow used for the design experiments is a simplified version of the flow described in detail in [38]. The flow achieves complete RTL-to-GDSII transformation for 3DICs[1].

The distinctive feature of this flow is that commercial single-tier (one layer of

[1]Thanks to Hao Hua who developed the original 3DIC flow and standard cell library

active devices) tools are used instead of pure 3D-aware tools built from scratch. The major advantage of such an approach is that the flow development was achieved much more quickly than if truly 3D-aware tools had been written. Also, mostly proven commercial technology is used instead of newer tools which are not well supported. Since the commercial tools available work only with single-tier designs, scripts have been written to patch up the outputs of the tools together to represent it as 3D data.

The main disadvantage of this approach is that some quality of the final output is sacrificed, which might not need to be sacrificed with true-3D tools. But in the interest of expediency in comparing equivalent single-tier and 3D designs and since no complete 3D-tools were readily available, we chose this approach. The expectancy is that the performance of any 3D tool which can be developed should be at least as good as the performance of this flow.

There has been good interest in researching 3DIC EDA tools in recent years. J.Cong et al. [21] developed an efficient thermal-driven floorplanning algorithm for 3DICs which allows block movement on the same-tier as well as different tiers. Balakrishnan et al. [9] developed a wire congestion and thermally aware 3D global placer. S.Das et al. [23] designed a 3D standard-cell-based placement tool, global router, and layout editor. Siozios et al. [64] developed a methodology for exploring and evaluating 3D FPGA architectures. Their tools include a 3D placer, router and power analyzer. Ababei et al. [7] developed placement and routing algorithms for ASIC and FPGA 3DICs. But neither of the tools support fabrication of real 3D ASIC chips. As far as we know, our 3DIC flow [38] was the first one to support 3D clock tree synthesis and

fabricate real chips with a standard cell methodology.

The next few sections will briefly explain the various steps in the flow.

## 3.2 Standard Cell Library

The target process used in creating the standard cell library is the MIT-LL 0.18um 3D-SOI technology process. This process uses a "via-last" approach [56], which means that inter-tier vias (ITVs) represent blockages to all routing layers. Therefore, along with the usual fundamental logic gates and Flip-flops, Inter-tier vias (ITVs) were created as part of the library. Creating an ITV is expensive in terms of silicon real-estate and putting two or more ITVs together usually requires less area than would be required if they were all created separately. Thus, some ITV standard cells were created such that they had 2 independent ITVs in them. An ITV connecting tiers A and B or B and C is composed of 2 standard sub-cells, each one to be placed in one of the tiers it is connecting. An ITV having connections to all 3 tiers (A,B and C) is composed of 3 standard sub-cells, each one to be placed in one of the 3 tiers.

Figure 3.1 shows the comparative sizes of some commonly used standard cells and the ITVs. As can be seen, an ITV connecting the Tiers B and C (Size:4X12$\mu m$ ) is slightly bigger than a minimum sized inverter (Size:3X12$\mu m$). The figure also shows a standard cell with 2 independent ITVs as part of the same standard cell. This cell uses the same area as the single ITV cell to create 2 ITVs. The ITVs shown are just one single-tier part of an ITV; the other part or parts will lie exactly on the top or bottom of another tier.

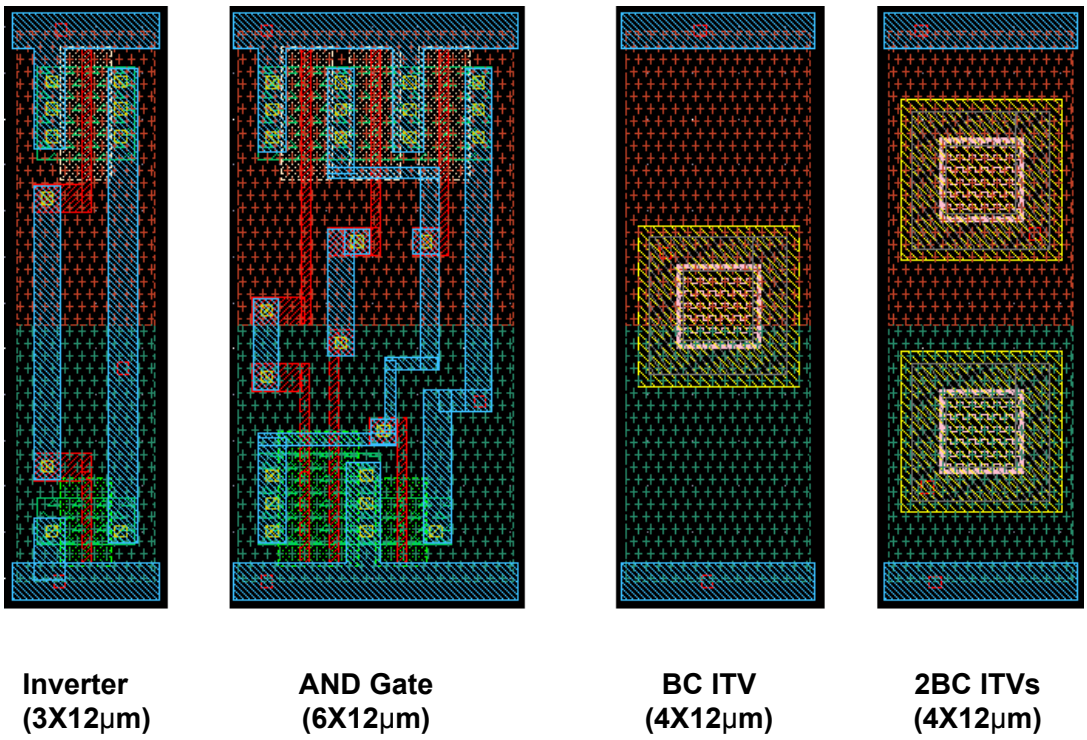


Figure 3.1: Standard Cells Relative Area Comparison

## 3.3 Memory Generation

The designs we experiment with need many memories of various configurations. In the absence of access to commercially designed memories, we had to develop our own methodology to generate the various memories needed. Since our intent was to design circuits whose performance could be studied by simulation, it was not needed to create actual memories which could be manufactured. As far as the rest of the design goes, generating black-box memory models rather than the complete layout was sufficient to get good estimates of system performance.

We initially simulated simple bit-cell layouts in the MIT Lincoln Labs technology

and extrapolated the results to larger memories. However, the results of this approach gave us memories that were several times larger and slower than what can be achieved in commercial $0.18\mu m$ processes. We wanted an approach that would allow us to predict the performance improvement from a commercial technology. Therefore, for each memory model designed, the appropriate parameters viz. bus width and storage capacity were used to get the memory characteristics from the CACTI [68] tool. According to its authors, CACTI is an integrated cache access time, cycle time, area, leakage, and dynamic power model and is primarily intended for use by computer architects to better understand the performance tradeoffs inherent in different cache sizes and organizations. Although it is mainly designed for cache structures, it also gives memory characteristics for SRAM memory.

For each of the memory that we use in our design, we get the area, aspect ratio, power, and access time estimates using CACTI for the particular configuration using the $0.18\mu m$ technology. These parameters are then used to generate the LEF (physical layers and dimensions) and LIB (functional timing) models for each memory.

Figure 3.2 shows the generic Memory structure for any SRAM memory. The row decoders and sense amplifiers physically lie onto two adjacent sides of the memory. The internal structure could be modified to create sub-banks, if they are needed to optimize the memory performance. As far as the system designer is concerned, the memory can be thought of as a black box with address lines and data busses. As will be shown in Chapter 4, the row decoders can be replaced with a shift register and smaller decoder logic to speed up the memory access time.

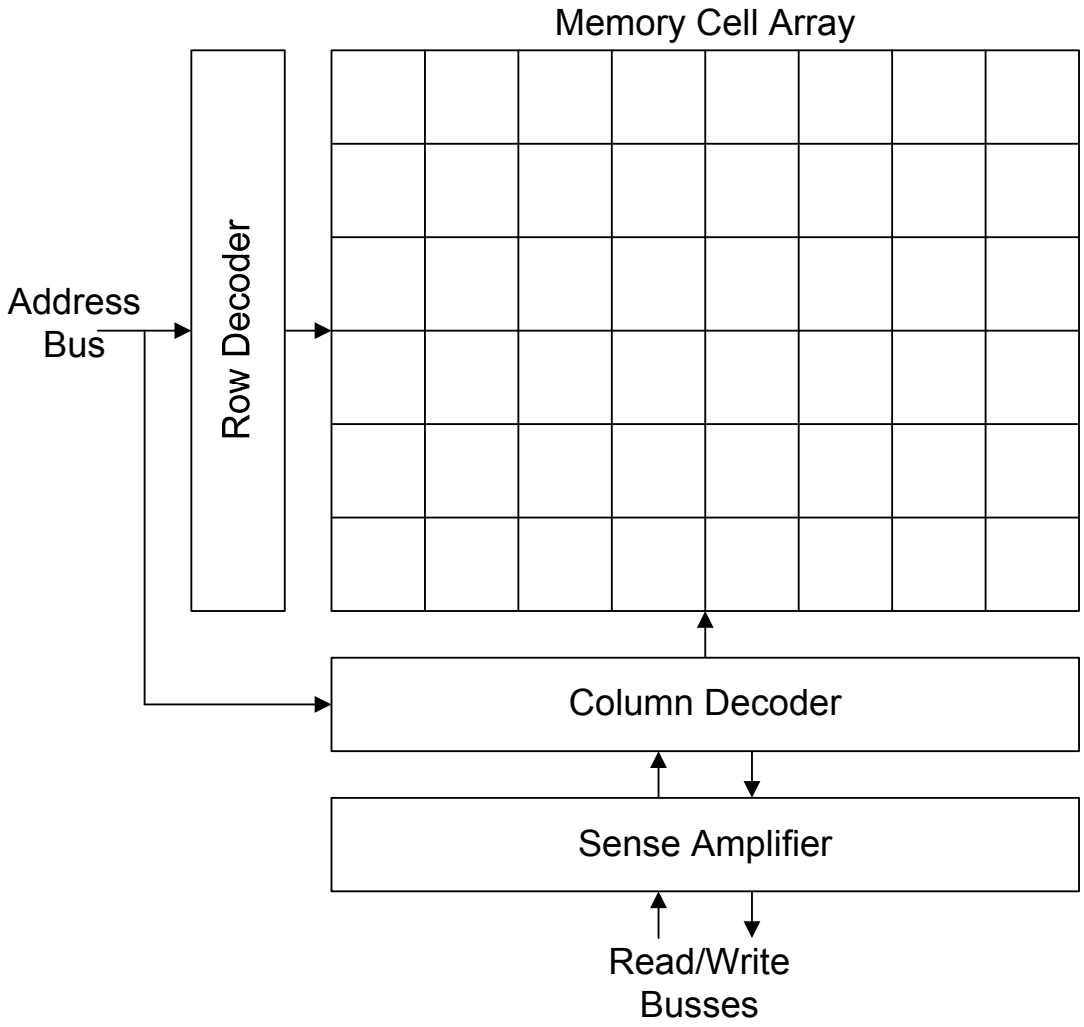


Figure 3.2: Generic Memory Structure

Figure 3.3 shows the structural representation of a generic dummy LEF file created for the 3DIC flow. We assume that the address lines for the memory are equally spaced on one side and the read and write data busses are equally spaced on its adjacent side. The read and write enable pins which would control the sense amplifiers were assumed to be at the center of the side with the data busses. That leaves the other

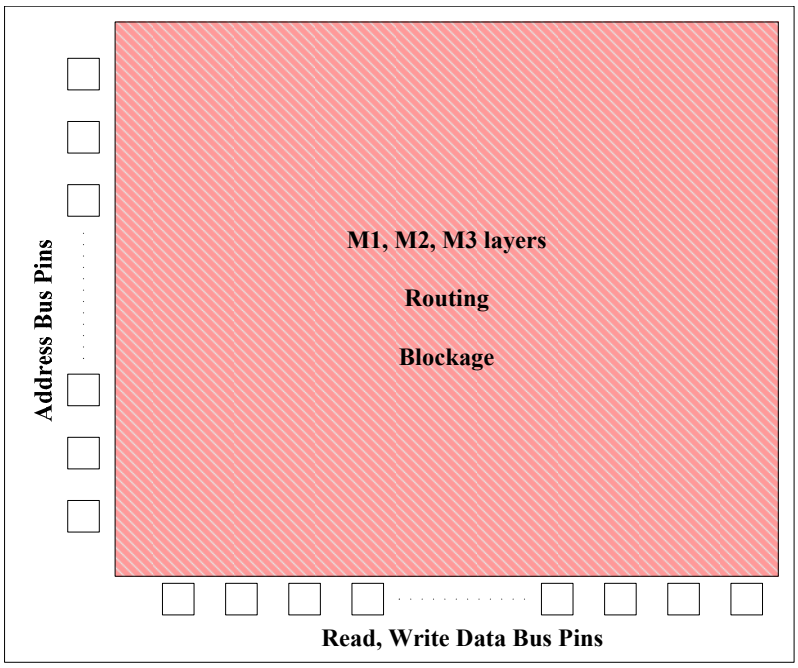


Figure 3.3: Generic Memory LEF file

two sides with no pins. For any given size of the memory, the area, aspect ratio, number of address lines and data bus width can be put together to get the distance between the pins.

Even though most of the memories are single-ported, they still have separate read and write data and address busses. Only one set of data and address busses will be active at any given time. The write and read enable pins will decide which set is active. The peripheral circuitry external to the memories has to make sure that at least one of the enable pins is inactive at any given time. In the case of double-ported memories, both the ports can be active at the same time.

Since memories usually do not allow any external wires to route above them, the

LEF models had blockages for all the 3 metal layers over the surface area of the memory. This forces the router to route any wires around the memory. The bigger the size of the memory, the bigger the blockages for placing other cells and routing resources.

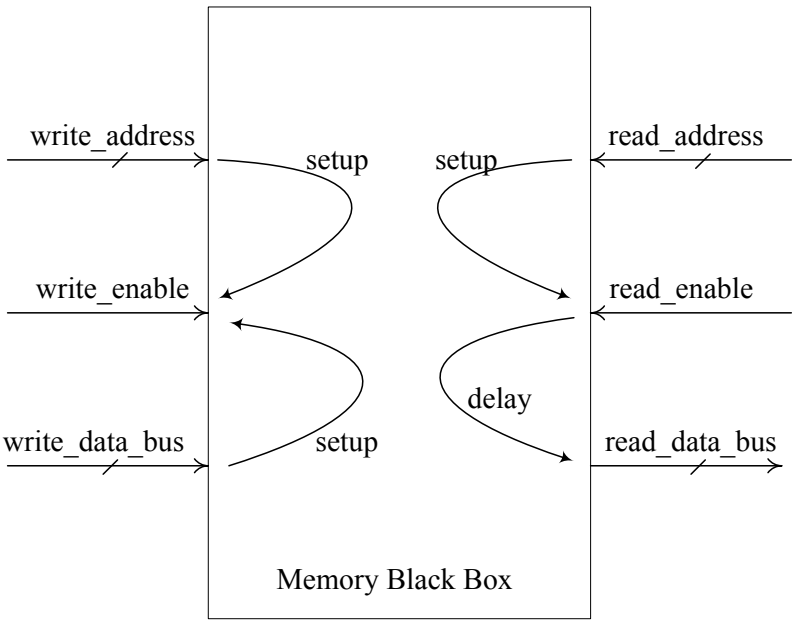


Figure 3.4: Generic Memory LIB file Timing Arcs

For timing analysis of any memory-based design using Synopsys Primetime, a timing model of the memory (LIB file) had to be generated. The access time obtained for a particular memory from the CACTI tool is used to denote the delay between the change in the read enable or read address and getting the memory read data at the read data bus. From the point of view of timing analysis, the memory can be thought of as combinational logic with a delay of this access time. To simplify creation of this model, some of the timing requirements like those regarding the write

enable signal and write bus were not modeled. For an asynchronous SRAM memory, the read operation generally requires more time compared to the write operation and is in the critical path. Hence only setup timing restriction on the read operation was modeled and deemed sufficient for timing analysis of the designs. Fig 3.4 shows the timing arcs modeled for the memories.

## 3.4 Steps

The 3DIC flow is similar to any usual single-tier RTL-to-GDSII EDA flow. The difference is that the design has to be partitioned into the number of tiers available (3 in our case) and each tier is placed and routed as if any other single-tier design, but with some inter-tier constraints.

Figure 3.5 shows a high-level view of the 3DIC flow. After the partitioning stage, each of the steps is a multi-tier version of the usual EDA flow. During each stage, the feasibility of the design is checked and if not found feasible, another iteration is done to tweak the design. Examples of non-feasible characteristics include overlapping cell placement, ITVs not aligned correctly between tiers, insufficient routing resources, etc.

The original 3DIC flow is described in detail in [38]. The following sections will explain some of the steps briefly and emphasize on the differences from the original flow.

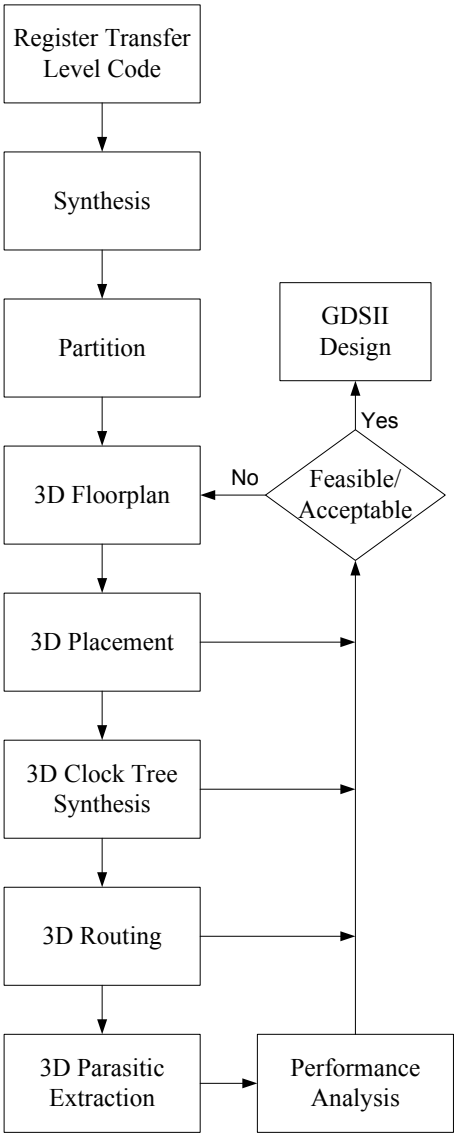


Figure 3.5: 3DIC flow

### 3.4.1 Synthesis and Partitioning

The input to the flow is Register Transfer Level (RTL) code as would be written in a hardware descriptive language (HDL) for a single-tier design. This is then synthesized using Synopsys Design Compiler with required timing constraints to get a

standard cell netlist.

This netlist is then partitioned using kmetis [47] from the University of Minnesota and 3 "independent" netlists are formed. Kmetis tries to reduce the cuts between the tiers, meaning that it attempts to minimize the number of inter-tier vias.

In case of the FIFO and FFT designs that will be described in later chapters, some manual partitioning of the memories had to be done. Separate RTL code was used for the three tier and single-tier versions to take advantage of three tier processing. Thus, the designs were synthesized differently to aid in the semi-automatic division between tiers. The rest of the unpartitioned design was processed by kmetis for partitioning into 3 tiers. The partitioner also tries to balance the areas of the 3 tiers while minimizing the number of cuts. Inter-tier vias (ITV) are added to the netlists of all the 3 tiers.

### 3.4.2 Floorplan and Placement

After the partition is done, the place and route flow is done separately on the individual tiers using the commercial single-tier tool SOC Encounter from Cadence. Once kmetis decides the partitions, no inter-tier transfer of cells is done. Such a step could potentially be performed by a true 3D-tool to get better results. For any generic design, it makes sense to pre-place the hard macros, if any, before the standard cells. For the FIFO and FFT designs, the memories are the hard macros and are pre-placed in an iterative process as described below. Figure 3.6 shows a detailed flowchart for the 3DIC EDA design flow.

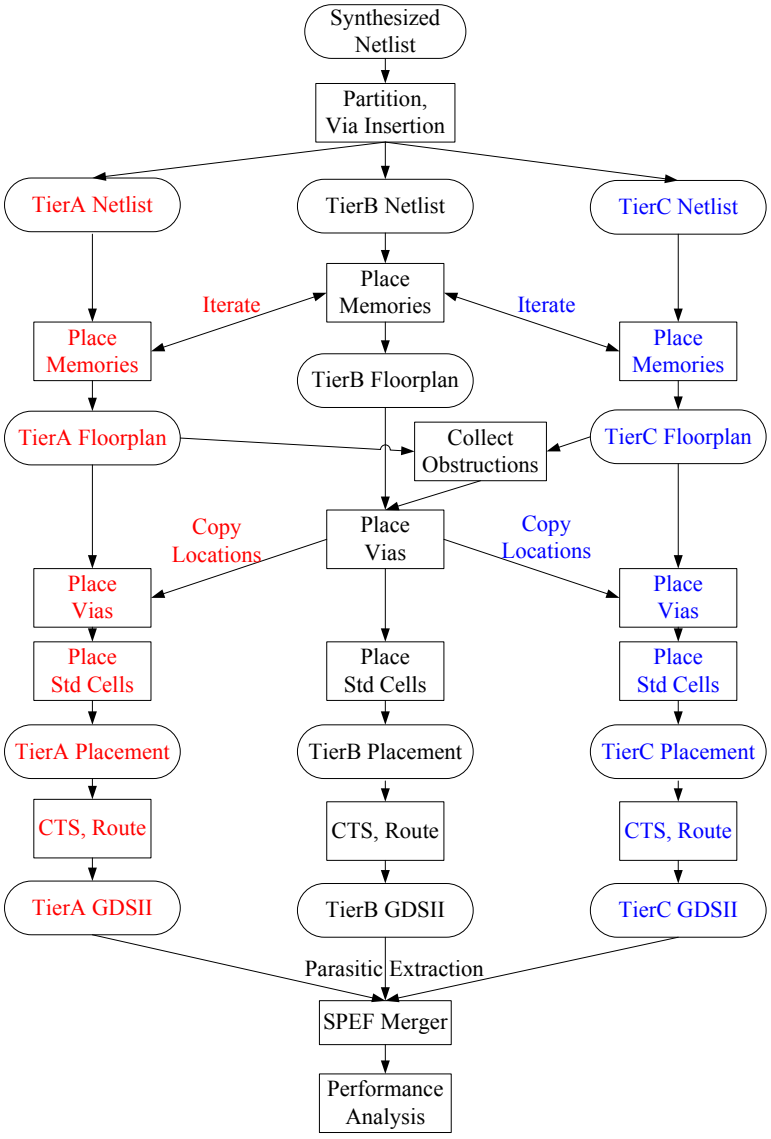


Figure 3.6: 3DIC flow in detail

An initial placement is done individually on each of the tiers using Encounter. There are no constraints given to Encounter during this step. The 3 placed designs are then manually studied and modified to get the next iterated placement. Modifications involve manually changing the locations of memories to get a better floorplan.

In our designs, some memories are split into 3 tiers and each of these parts is supposed to be placed at the exact same locations in the 3 tiers. To accomplish this, a script is executed which finds the locations of such memories in TierB and creates constraints to place the corresponding parts on Tiers A and C respectively.

After the memories are pre-placed, it is the turn of the ITVs to be placed. Since every ITV has to have one subcell in TierB, we complete the ITV placement step for TierB and just copy the corresponding ITV subcell locations to Tiers A and C. As the ITVs cannot be placed at any of the locations where a memory is already placed, constraints have to be created which prevents this scenario. A list of blockages due to the placed memories is created. The TierB design with placed memories is loaded, these blockages due to memories from other tiers, if any are applied and the ITVs for TierB are placed, along with the rest of the TierB netlist. This becomes the final TierB placed design which will be given to the routing stage.

The TierA design with pre-placed memories is then loaded and the ITV constraints corresponding to TierB's placed ITVs are applied. The rest of the TierA netlist is then placed and saved to be given as input to the routing stage. The same procedure is followed for TierC and we get three independent placed designs as output of this stage.

### 3.4.3 Routing

The routing stage is split into special nets (clock and reset) routing and other signal routing. The placed design for each tier is loaded individually and the clock

and reset trees are first synthesized independently. The starting point for each of the clock trees in the three tiers is at the exact same spot for all tiers. These starting points are shorted with an TierA-B-C ITV. The same is done with the reset trees.

Next each of the tiers is independently routed and parasitics from the design are extracted. All the three tiers are read into Primetime or PrimePower to get timing and power results for the entire design.

## 3.5 Performance Analysis

Synopsys Primetime is then used to get the timing reports from the unified netlist and the 3 individual SPEF (parasitics) files. Similarly, Synopsys PrimePower is used to get the power reports for the design. The energy dissipated for the operation is then calculated.

## 3.6 Limitations of the flow

There are certain aspects of a real 3D chip which we have not considered so that we could simplify our original 3DIC flow. We were primarily interested in improving the timing of our design and any aspect which did not have a huge impact on timing could be ignored.

For real designs, proper power rings would have to be designed around the core as a whole and around the various memories. Specially designed pads for the I/Os will have to be placed at proper locations with tier restrictions if any.

### 3.6 Limitations of the flow

As this is a 3D design, thermal vias will have to be inserted at hot spots in the chip. They will lead to more routing blockages and reduce performance of the router. Depending on the technology and design, there could be restrictions on how big can an area be without having thermal vias. This could put restrictions on how big single memories can be and on which tier they can be placed. We leave thermal analysis for designs with memories as future work.

# Chapter 4

# FIFO Design

This chapter presents a 3DIC case-study [66], in the design of First-in First-out (FIFO) buffers.

## 4.1 Introduction

3DICs can potentially improve the performance of integrated systems by bringing large amounts of memory close to processors, increasing speed and reducing power. First-in First-out (FIFO) buffers are used to store data in memory and read it in the same order that was stored. Such buffers are used in wide-ranging applications like real-time DSP systems, telecommunications, biomedical imaging, packet buffering, bus width matching, frequency coupling and many others. There are a significant number of FIFO structures in pipelined FFTs and any improvement in the FIFO performance ameliorates the FFT performance. We focus on generic FIFO structures

in this chapter and note that the FFT will require a specific version of the FIFO known as a delay buffer. Such a delay buffer will be a stripped-down version of the generic FIFO and hence easier to build.

When required storage capacity of the FIFO is sufficiently small, a shift register [45] or register file [31] made of Flip-Flops can be used to implement them. Otherwise, large memories of RAM type [33] can be used to build FIFOs. The memory accesses can be controlled by special write and read pointers that increment when data is either written or read into the memory. Thus, the address decoders present in conventional Random Access Memories can be replaced with simpler shift registers that control the Column and Row Enables for the memory, to model serial accesses of the FIFO. Specialized address pointers have been previously developed [69] for controlling the read and write accesses to the FIFO storage memory.

To provide stall-free access to the FIFO memory, it conventionally has dual-port access [30, 29, 62], thus allowing writes and reads to happen simultaneously. Dual-ported memory is generally slower than corresponding identical capacity single-ported memory and also typically increases the area requirements by 33% compared to the single-port memory [70]. Increasing number of ports also leads to increased power consumption. Thus, it would be helpful to use single-ported memories to design FIFO structures.

In order to provide a fair comparison of 3DIC FIFOs to traditional single-tier FIFOs, we have developed a high-performance FIFO architecture that uses single-ported memories and designed it in the MIT Lincoln Lab $0.18\mu m$ three-tier fully

depleted silicon on insulator (FDSOI) process [1], in both single-tier and three-tier implementations. In order to perform the study quickly, a standard-cell methodology was used to implement the control and decode logic, with a custom layout assumed for the bit-cell array.

The rest of the chapter is organized as follows: Section 4.2 looks at a multi-banked structure that can be used to design FIFO buffers. Section 4.3 proposes a novel way of designing address pointers which saves area and power compared to conventional approaches. Section 4.4 describes the design of a 2KB and 16KB FIFO and compares the single-tier and 3D versions.

## 4.2 Memory Architectures

FIFO buffers are constructed using memory when the required storage space makes it prohibitively larger to use Flip-Flops for building it. The buffer memory conventionally has dual-port access [30, 29, 62], to permit write and read accesses to the FIFO to occur simultaneously. The accesses to the FIFO are stall-free, as long as the memory capacity is not filled.

Fig. 4.1 shows a typical way of designing a FIFO using double-ported RAM memory.

Since the memory accesses of a FIFO are not random, the address decoder logic is not required and can be replaced with simple one-hot shift registers to generate the row and column enables (representing the read/write pointers) to access required memory locations. The FIFO Controller receives the Write Enable and Read Enable

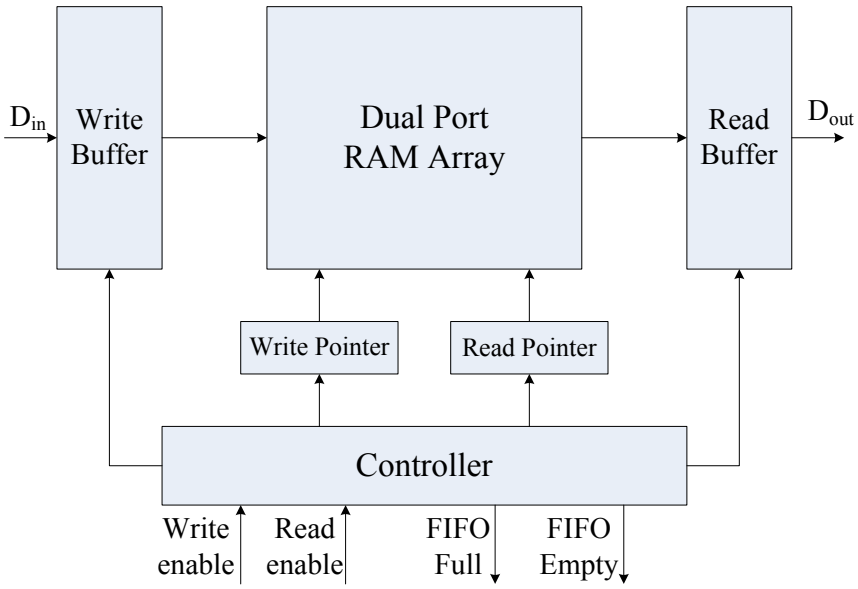


Figure 4.1: Conventional Dual-port RAM FIFO

signals and logically increments the Write or Read Pointer as required. The Read pointer follows the Write pointer so that the data that was previously written is read back in the same order, thus fulfilling the criteria of a First-in First-out Buffer.

The controller also generates the FIFO Empty and Full Status flags depending on how much of the FIFO memory is filled.

Pipeline FFTs use plenty of delay buffers and hence we are interested in the operation of these special FIFO structures. Conceptually, the operation of a delay buffer is as shown in Fig.4.2. The input data will be written in sequential fashion beginning from memory location 0 to memory location N-1. The output data bus can read the exact same locations in the same order, during the same clock cycles. To prevent overwriting of new data before it is being read, the read data bus should be

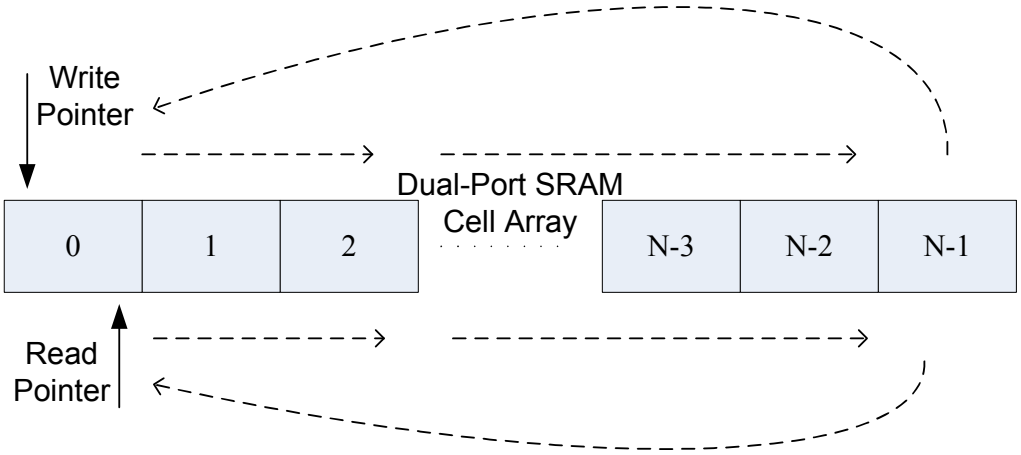


Figure 4.2: Dual Port SRAM used as a Delay Buffer

driven by the memory on the positive edge of clock and the write data bus should be sampled by the memory on the negative edge of the clock (or vice versa). Thus, this simple delay buffer scheme not only uses a dual-port memory (thus increasing the size of the memory by up to 33% over single-ported memory [70]), it also requires dual edge access. This forces the clock period of the entire system to be at least equal to the sum of the read and write access times of the memory. Dual-ported memories have been used to design FIFO structures by Feldman and Duzer [29], Shibata et.al [62] and Kanopoulus and Hallenbeck [46].

Since dual-ported memory is typically larger, slower and consumes more power compared to single-ported memory, it is beneficial to use single-ported memory for the design of FIFOs.

The fundamental idea to avoid requiring a double-ported memory is to take advantage of the certainty of the order of memory addresses that are accessed by the

FIFO buffer. The FIFO controller can prefetch or cache the next memory read access into a read buffer when a write access is not taking place. This can be easily achieved by multi-banking the available storage memory into 2 or more smaller memories, the combination of whose capacities would give the capacity of the original double-ported memory. The memory accesses can then be interleaved, thus simultaneously utilizing one bank of (single-port) memory for writing and the other for reading, if the corresponding read buffer is not filled. A bypass mechanism can be used to fill the read buffer directly from the write port, when the FIFO is empty. This approach lets the allowable clock period to be the time for one memory access.

Fig. 4.3 shows the schematic for such an arrangement. The write and read buffers and the bypass mechanism shown are independently present for each of the memory banks. The Controller uses the Write Demux to route the data being written into the FIFO to the proper memory bank. During each cycle, a maximum of one of the $N$ banks is written while the others can take this opportunity to refill their individual read buffers if needed. The Read Mux routes the proper Memory data to be read to the output port in the same order that it was written. This procedure allows the use of single-port memories. The advantage of increasing the number of banks is that each memory has to be of smaller size and hence will be faster. The disadvantage is that it increases the size of the input write Demultiplexor and the output Multiplexor and would cause an increase in congestion and the length of interconnecting wires.

Memory banking and interleaving structures have been widely used [25, 48] to construct serial access memory.

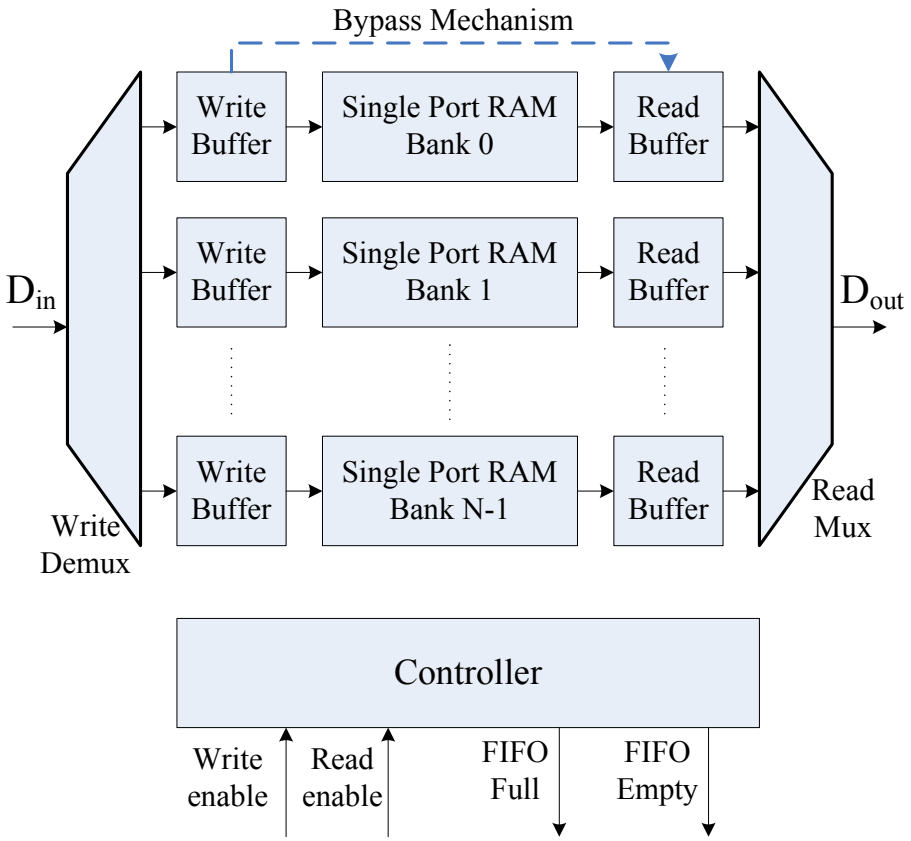


Figure 4.3: Multi-Banked Interleaved Single-port RAM FIFO

In the case of a delay buffer, even the bypass mechanism is not required. The read address should just follow the write address at a fixed distance. This is usually accomplished by keeping the read and write address the same and the capacity of the buffer as the delay required. We can change this scheme slightly by making the read pointer to be always one more than the write pointer. In this manner, the memory can be split into two and interleaved. Each of the memories then needs to be just single-ported. Since the read address is now one cycle ahead of what it should have been, a Flip-Flop delay stage can be added to delay the value coming out at the

output of this FIFO. This scheme as shown in Fig.4.4 is similar to the ones used by He and Torkelson [35] and Li and Wanhammar [51].

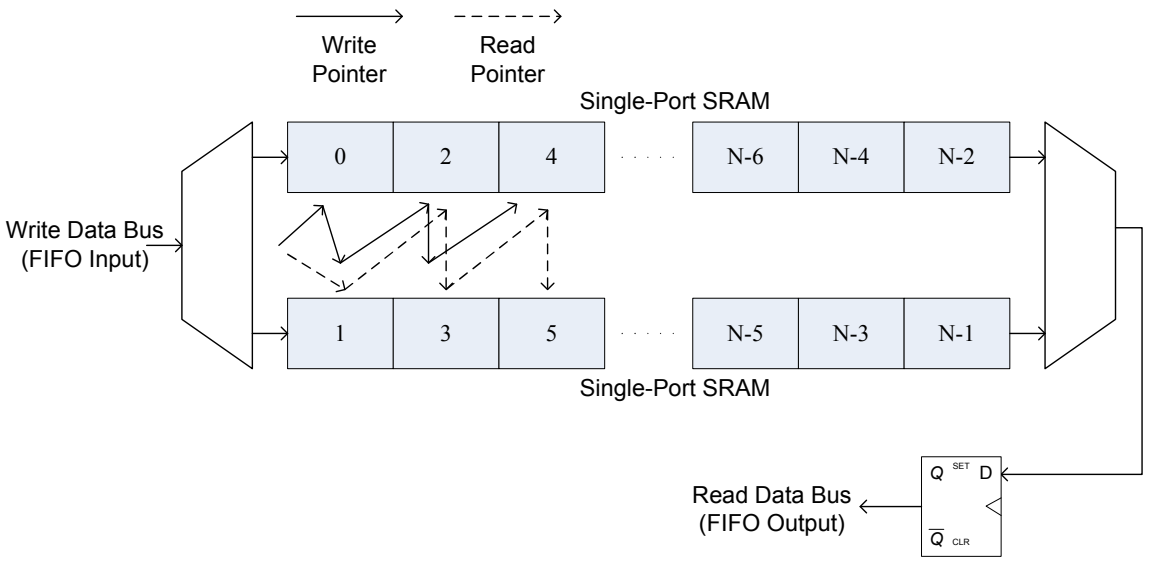


Figure 4.4: 2 Single Port SRAMs used as a delay buffer

Since we are using 3DIC technology, we can put these 2 memories into different tiers to reduce power density and access times to each.

Taking this notion further, we can split the memory into 3 submemories, to take advantage of the MIT 3DLL process, which has 3 tiers. In this scenario, the read and write pointers can point to the same logical address in the conceptual FIFO, but they would be totally different physical addresses. The read pointer can then follow the write pointer with a fixed delay of N cycles in a round robin fashion. This is conceptually shown in Fig.4.5. As none of the FIFO sizes in our design is divisible by 3, we can be certain that never would the read and write pointers point to the same tier and hence we are free to use single-port RAMs.

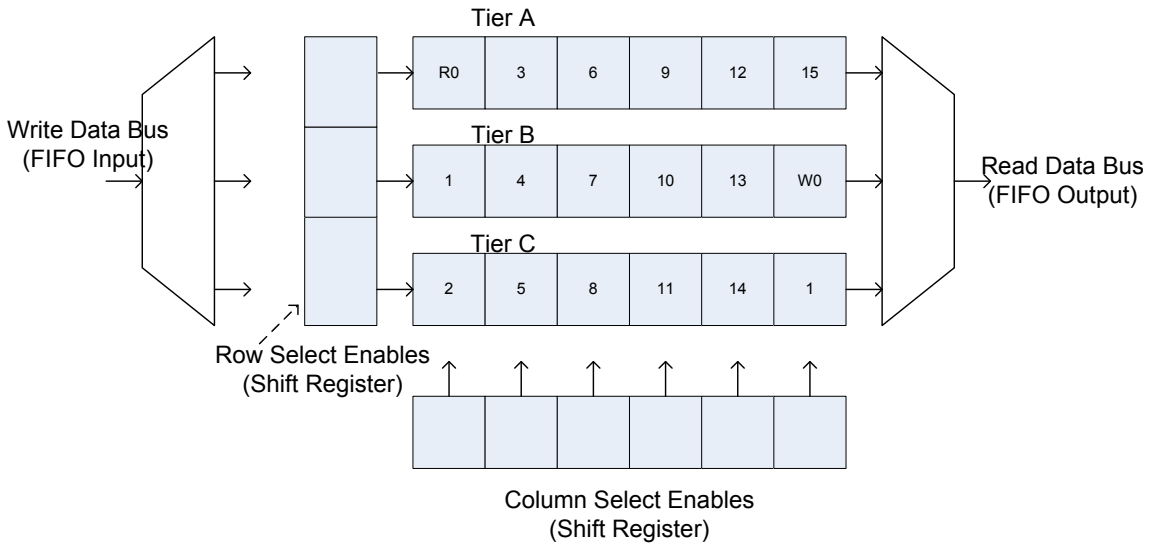


Figure 4.5: 3 Single Port RAMs used as a delay buffer

Since our application is such that the read and write pointers are always at a fixed conceptual distance from each other, we don't need flag generation logic to indicate when the FIFO is empty or full. Also the address generation for the pointers is done by row and column enable signals which can be generated from specialized shift circuitry.

To further improve the performance of the system, the width of the data ports of the internal memory ($W_{mem}$) can be made wider than the width of the FIFO ports ($W_{fifo}$). Internal read and write buffers of width ($W_{mem}$) would be required to store the data transferred between the memory and FIFO ports Then, if $W_{mem} = K * W_{fifo}$, the FIFO can theoretically run at $K$ times the speed at which the Memory can be accessed. Fig.4.6 shows a possible method to wide port each memory bank and potentially increase throughput by $K$ times.

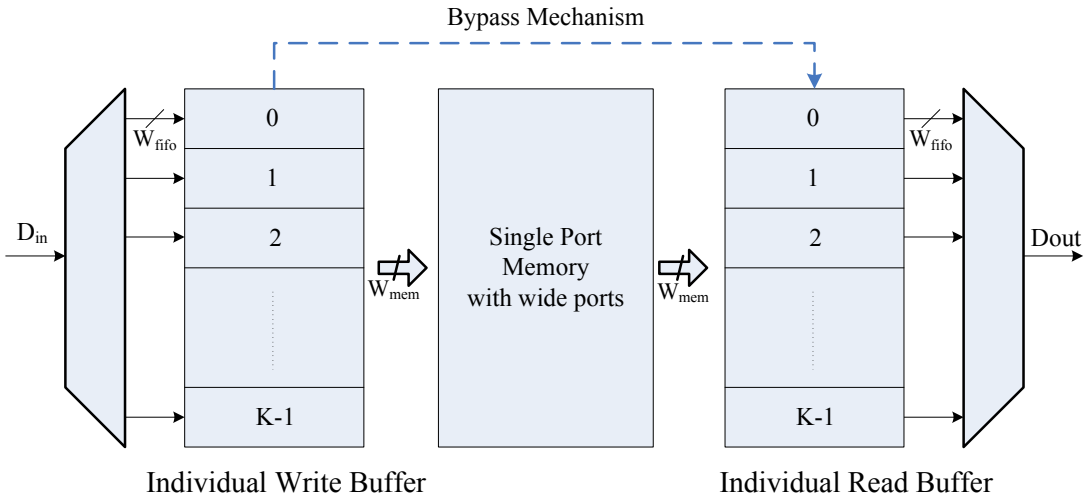


Figure 4.6: Buffering each bank access using memory with wide busses

Interleaved banking structures are well suited for 3DIC Technology as the different banks can be placed on top of each other, thus reducing the interconnect distances between the controllers, buffers and read/write ports of the different memories. A first-order performance analysis of large 3D SRAM memory used as a cache by Zeng et.al[75] predicts cache latency improvements up to 50% compared to corresponding 2D memory for sizes of 16 Mb. In our case, interleaving creates 2 or more independent sub-FIFOs of smaller size than the original FIFO. All the sub-FIFOs are guaranteed to have at most one read or write access per cycle, thus allowing single-ported memories to be used.

## 4.3 Design of Address Pointers

As indicated before, a FIFO does not require complete address decoding logic as the memory accesses are not random. The simplest way to generate the predictable addresses is to use ring register chains for the row and column enables as shown in Fig.4.7. The box shows some of the individual cells of the memory data array which are enabled to be accessed based on which of the row/column enables are 1. For every

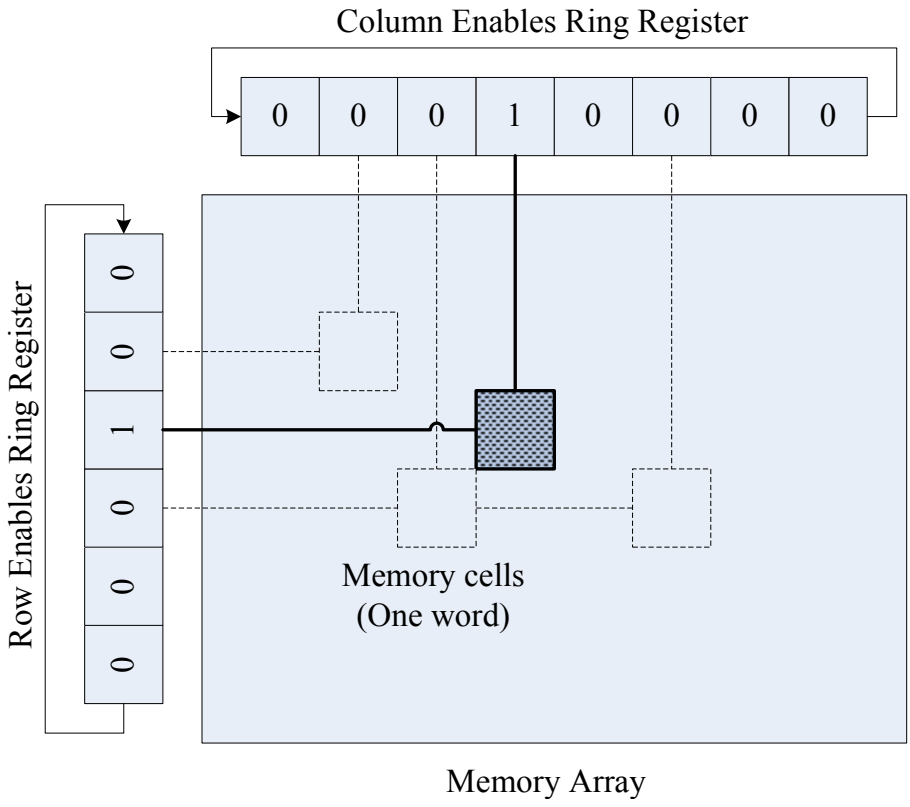


Figure 4.7: Address Pointer Generation using Ring Registers

iteration that the column one-hot ring register completes, the row ring register shifts by 1 or vice versa. In this example, the cell selected to be accessed next is highlighted

as a black box.

But using a ring-chain would require a long return path from the last D-FF in the chain to the first one. The capacitance on this path would cause the timing to deteriorate, if the size of the chain is large enough. In a custom FIFO, this problem is solved by interleaving the return-path of ring-chain with the forward-path, thereby keeping these wires short and out of the critical path. However, it is difficult to ensure this interleaving in an automated standard cell based place-and-route methodology. Therefore, to mitigate this problem, we use a shift register chain which is not tied together in a ring pattern. The entrance to the shift register chain is fed through a counter which generates a high pulse of one clock cycle after every $N$ number of write enables (or read enables) are received by the FIFO, where $N$ is the length of the shift register chain. There will be 2 such generators for writes (row and column) as well as reads, giving a total of 4 generators. The length of the chain for rows and columns may be different depending on the structure of the memory array.

To reduce the number of Flip-Flops in these chains, we take advantage of the fact that the output we require for the column (or row) enables is one-hot. In our proposed scheme, to generate a one-hot output from a register chain of length $N * M$, we use 2 register chains of size $N$ and $M$ each, and have a single-layer decoder of $2 * N * M$ 2-input NOR gates. Such kind of a logical arrangement is shown in Fig.4.8. Both the shifter chains have a single Flip-Flop having an output of 0 and the rest have an output of 1. The matrix nature of the 2-input NOR gates ensures that only one of them has an output of 1 and the rest have an output of 0. In this way, we can

conceptually use only one layer of the original address decoder to reduce the number of Flip-Flops required from $N * M$ as in conventional approaches [69, 62] to $N + M$. The added penalty is of $2 * N * M$ 2-input NOR gates. Each of the NOR gates could also be replaced with a pass transistor mux, thus reducing the area penalty even further.

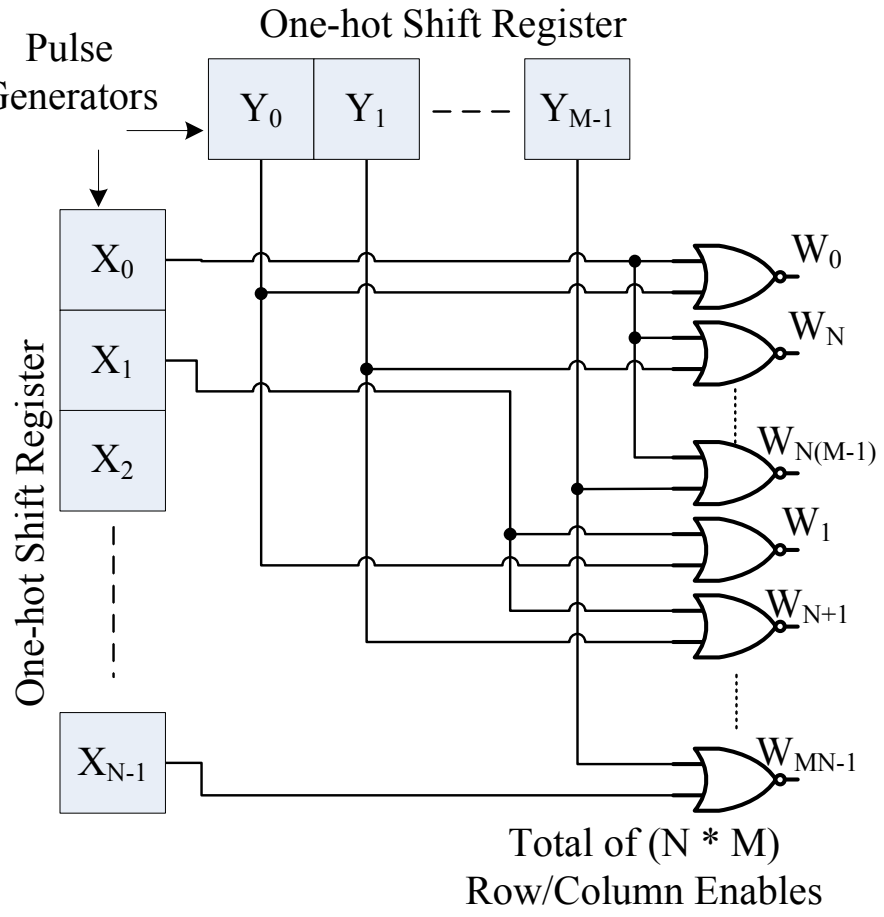


Figure 4.8: Generic design for column and row enable generators

As can be seen in Fig.4.8, the enables ($W_L$) generated are not in physically contiguous order. But, since the only requirement of a FIFO buffer is that the data that

is written should be read in the same order, it does not matter in what order they are actually stored inside. When identical circuits are used for both the write and read address pointers, it suffices the criteria of the same order of accesses.

A disadvantage of this approach is that fanout for the X registers is $M$ and the fanout for the Y registers is $N$. This would also lead to long wires and hence large capacitance. Hence, it would make sense to reduce $M$ leading to reduction of congestion. We use one Flip-Flop for the Y chain and generate the required signals from the $Q$ and $Q_{bar}$ outputs of this Flip-Flop($Y_0$). Even though these signals have to drive $N/2$ NOR gates each, the one-hot nature of the chain means that only the generation of $W_0$ and $W_N$ (when either of them changes from 0 to 1) needs to be done in one cycle and could be in the critical path.

To illustrate this, consider a case where 256 row enables have to be generated using a shift register of 128 Flip-flops and 256 NOR gates. As shown in Fig. 4.9, only a small buffer tree ($B_0 - B_3$) is used to drive $Y_0$ and $\overline{Y_0}$. The outputs of $B_2$ and $B_3$ have a very high fanout and are slow nets. The critical path of this circuit can be sensitized as shown in the figure. The transitions in the values of all signals are indicated in the form $0 \rightarrow 1$. $W_{127}$ goes from 1 to 0 and simultaneously $W_{128}$ has to become 1 to indicate the next row of the memory to be accessed. For this to happen, $X_0$ and $\overline{Y_0}$ both have to go from 1 to 0. Since they both have a low fanout, and are close to the source flip-flops, that is achieved in a small time. $B_1$ can also quickly transition to 0, but $B_3$ cannot. This is not a problem though as all outputs below $W_2$ are going to stay at 1 since $X_2 - X_{127}$ are all 1. $B_3$ comes in the critical path

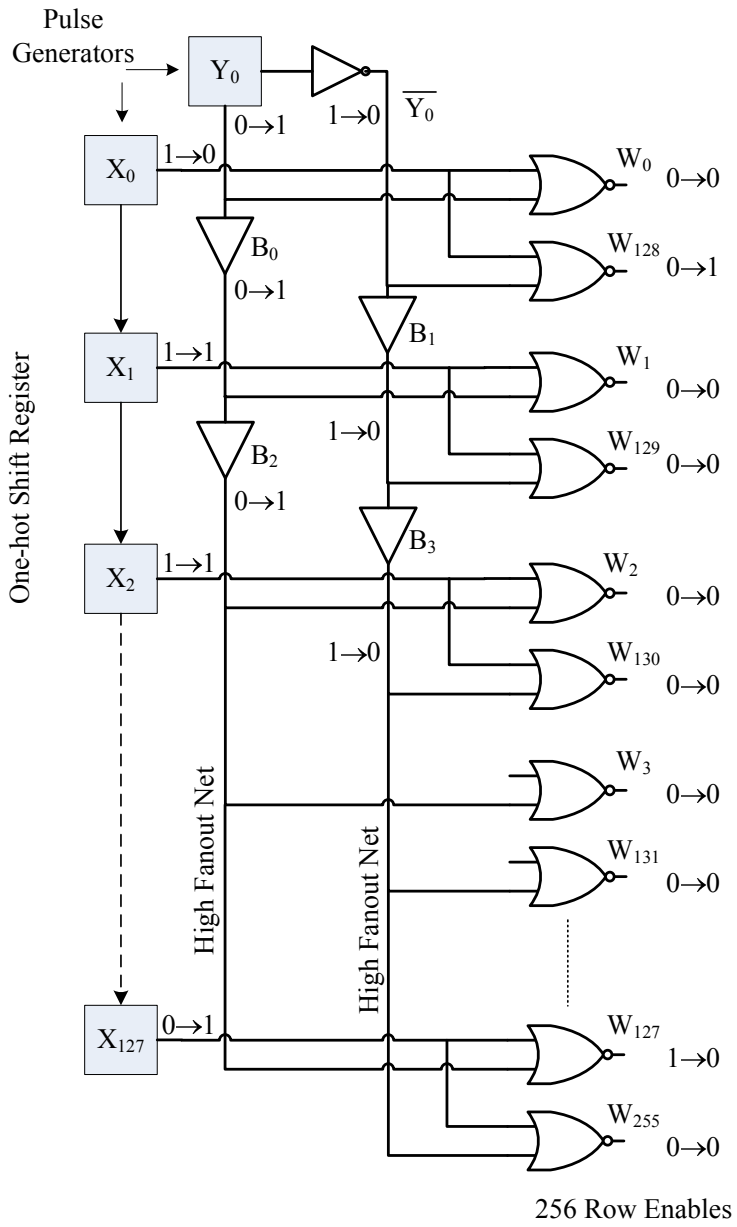


Figure 4.9: 256 Row Enables Generation using Proposed Method

only when $X_2$ becomes 0, which should take at least 2 more clock cycles. In fact, due to the nature of the 3D design, since writes (or reads) round-robin between banks, it would take $X_2$ at least 6 cycles to become 0. Thus $B_3$ can take 7 cycles instead of 1 to

transition. Due to the multi-cycle nature of this path, $B_3$ can be slow and the design would still work. For bigger designs, the buffer tree can be extended further till it meets the critical timing, taking into consideration this multi-cycle path. Hence, the proposed changes do not need a high power buffer tree to drive $Y_0$ or $\overline{Y_0}$, thus saving power.

## 4.4 Design of Proposed FIFO

We design two FIFO buffers of 2KB and 16KB storage capacity using $0.18\mu m$ technology in different configurations and demonstrate the advantage of our method. The figures used in the next paragraphs relate to the 16KB FIFO.

The first configuration (as shown in Fig.4.1) uses a conventional 16KB double ported SRAM array for the FIFO design. The second configuration uses 2 banks of 8KB of single-port memories on a single tier (2D design). Such a configuration is as shown in Fig.4.3 with 2 banks of memories.

The third configuration (as shown in Fig.4.10) uses 3 banks of 5.5KB bits of single-port memories on 3 tiers of the MIT FDSOI process [16]. As seen in Fig.4.10, each of the 3 Tiers A, B and C have a write buffer, a read buffer, a single-port memory and an address generator like the one described in Section 4.3. The input Mux, the output Demux and the controller are on Tier B and connected to the blocks on Tiers A and C using inter-tier vias. As the major portions of the design on each tier are identical and are placed right on top of one another, the lengths of the wires between different tiers is quite small. The critical path of the design lies through the memory and

the maximum delay is during the read cycle of the memory. We use the 3DIC flow described by Hao et.al[39] to analyze this configuration.

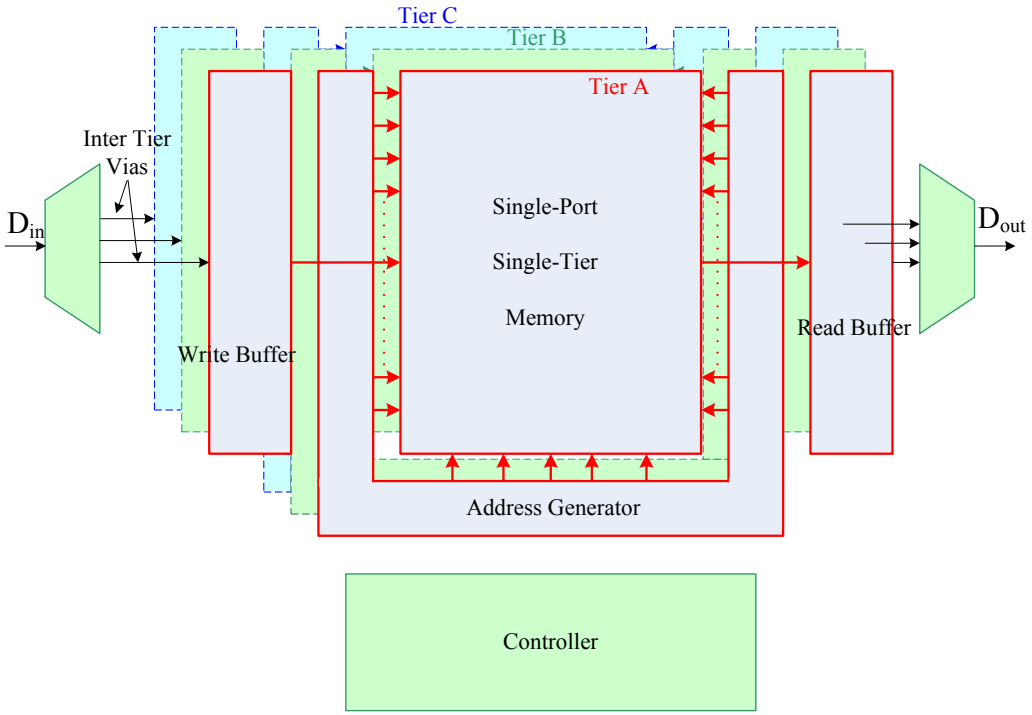


Figure 4.10: Three Tier FIFO Design

In each of the three configurations, we use the same standard cell library to synthesize the design and get the timing, area and energy estimates. Timing, area and energy estimates for the various memories were obtained from the CACTI [68] tool from HP for the $0.18\mu m$ technology node. Since the memories in our design do not have address decoders, the corresponding values for timing, area and energy were subtracted from the SRAM estimates given by CACTI. Dummy physical memories having the aspect ratio and area reported by CACTI were created and used in the ASIC Design flow.

## 4.4 Design of Proposed FIFO

During the standard cell synthesis of the design using Synopsys Design Compiler [3], we constrained the arithmetic critical paths in the design (incrementing/decrementing the counters) to be lesser than the critical path through the memory. For the sizes of the memories and the arithmetic components used, a custom design flow would have very easily met this condition. The synthesized design was then placed, routed and parasitics were extracted using Cadence Encounter [1]. The timing was then analyzed using Synopsys PrimeTime [5] and power using PrimePower [4].

Table 4.1 summarizes the analysis for the three configurations for the 16KB and 2KB capacity FIFOs. The minimum clock period to access the FIFO without timing violations is indicated. The energy shown is that which is consumed for one write and one read to the FIFO.

Table 4.1: Analysis

| 16KB FIFO | Area ($\mu m^2$) | Timing (ns) | Energy (nJ) |
|---|---|---|---|
| Dual Ported Memory | 6903000 | 1.84 | 0.0799 |
| Two Banked Single Tier | 2100992 | 1.34 | 0.0718 |
| 3DIC Three Banked | 2116800 | 1.23 | 0.0676 |

| 2KB FIFO | Area ($\mu m^2$) | Timing (ns) | Energy (nJ) |
|---|---|---|---|
| Dual Ported Memory | 852800 | 1.28 | 0.0459 |
| Two Banked Single Tier | 476100 | 1.21 | 0.0451 |
| 3DIC Three Banked | 573300 | 1.17 | 0.044 |

As shown in Table 4.1, the conventional architecture with Double Ported Memories is the slowest and the 3DIC architecture is the fastest (improving by 33% for 16KB

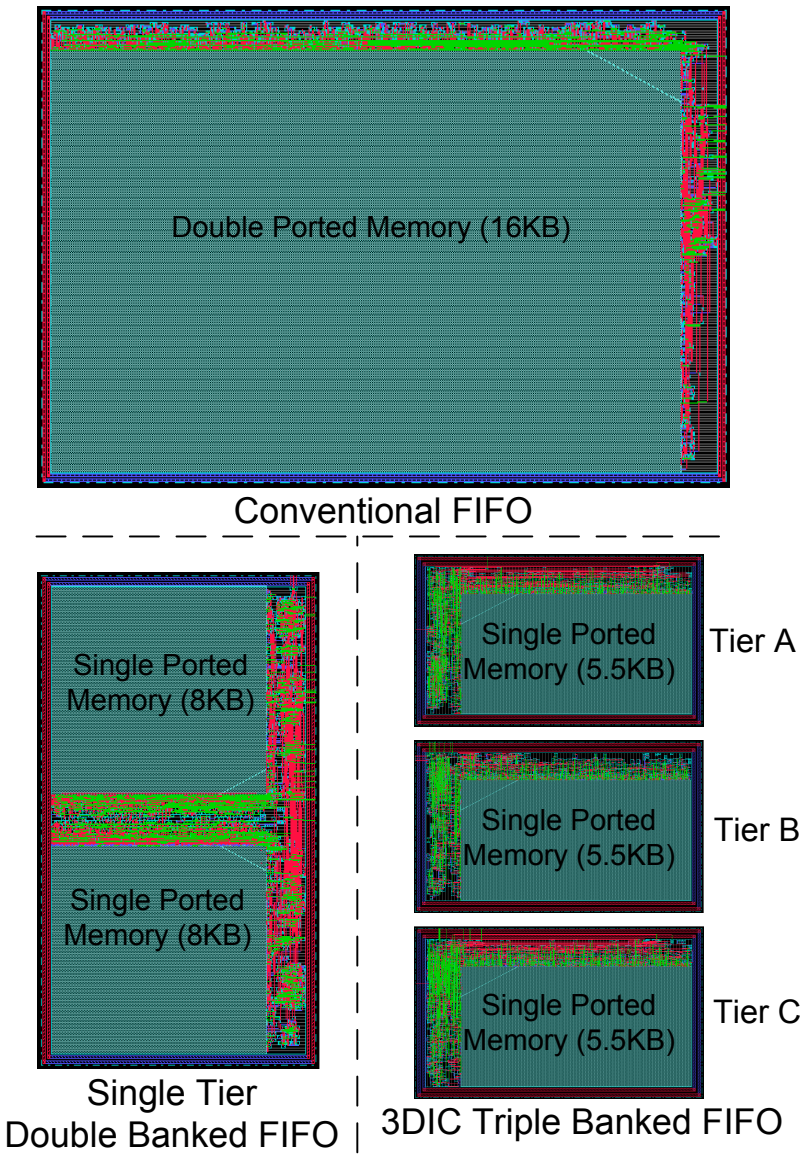


Figure 4.11: Relative layouts of various configurations for 16KB FIFO

FIFO). Similarly, the 3DIC architecture can be seen to be the most energy-efficient among the three (better by 15% for 16KB FIFO). The trends are more pronounced for the bigger FIFO. The 16KB FIFO is be seen to be operable at 800 MHz.

A 2KB custom-designed FIFO recently published [62] dissipates 46mW at 100MHz

in the 0.6$\mu m$ technology at 3.3V. Using the constant electric field scaling method [61], this design can be roughly expected to dissipate 5.32mW at 330MHz using a supply voltage of 1V in the 0.18$\mu m$ technology. That translates to energy consumed being 0.017nJ for one read and write access (in the same cycle). Taking into consideration that our designs are standard cell-based, operate at 1.8V and the memory parameters are rough estimates from CACTI, their performance is comparable with this design. The 3DIC technique can build upon the custom-design improvements in that case as well.

Figure 4.11 shows the physical layouts and relative sizes of the 3 configurations. The layouts are drawn to scale so it is easy to compare the areas. In each configuration, most of the area is occupied by the memory array. Also seen is the supporting logic made of standard cells. The area improvement of this technique is much larger than expected. This is due to the fact that the data array area for one 16KB dual ported memory reported by CACTI is nearly 5 times larger than the cumulative area of 3 single ported 5.5 KB memories for the 0.18$\mu m$ process. But an accurate estimate of the area would still not alter the timing and energy trends, which are of interest here.

## 4.5 Conclusion

We presented an architecture to design First-in First-out Buffers using 3DIC Technology. Usage of a 3DIC process resulted in 33% improvement in timing and 15% improvement in energy consumed compared to a double-ported memory based design

in single-tier process for a 16KB FIFO. The corresponding improvements compared to a single-tier double-banked design were 8% in timing and 6% in energy. The benefits of designing using 3DIC Technology are amplified when size of the FIFO becomes bigger. A method for designing the address pointers for the FIFO memory was also shown such that they use roughly half the number of Flip-Flops used in conventional designs.

# Chapter 5

# FFT Design

This chapter discusses the design of the proposed FFT processor in detail.

## 5.1 Proposed FFT Architecture

The proposed FFT design uses the Radix 2/4/8 Algorithm and the MDC Architecture. The goal is to demonstrate the advantage of using 3DIC technology for large-point FFT designs. The proposed design utilizes FIFO Buffering and Memory Interleaving to achieve clock frequency targets for large-point FFTs that would be difficult to achieve with 2D Technology and non-interleaved memories. A good advantage of using 3DIC Technology is decreasing overall interconnection length and routing congestion. Also, due to the additional silicon area available, additional SRAM can be had on-chip than would have been feasible in the 2D case.

The MDC architecture needs about 1.5 times more memory than the SDF ar-

chitecture for an equivalent point FFT, but can be two times faster than the SDF architecture. Our target is improving the throughput of the FFT processor and hence we choose the MDC architecture.

The disadvantage of using higher radix FFTs is that complexity of the control logic increases drastically once the size of radix goes above 8. Also, systems having a radix R requires R complex words to be fed to the butterfly at the same time and R complex words of outputs of the butterfly to be read at the same time. This stresses the memory bandwidth requirements and requires memory buffering schemes that slow down the system. Although higher-radix systems in general tend to require smaller number of arithmetic units, the added complexity and memory bandwidth requirements negate that advantage.

Approaches like those taken by Pronita Mehrotra [57] help substantially to increase the memory bandwidth with certain packaging methods, but the complexity factor is still important to consider. Also, although the number of multiplications decrease (with a diminishing return) as the radix increases, the number of additions actually increases after a certain point [8, 18]. This is illustrated in Table 5.1.

The processing elements for the Radix-2/4/8 Butterfly are as shown in Fig.5.1. They are modified from the single-delay-feedback versions shown in Jia [41] and Lin [52].

Complex multiplication by $\sqrt{2}/2$ can be done using 4 real multiplies and 2 additions. But when using fixed point numbers, real multiplication of a number by $2^n$ can be calculated by appropriately right-shifting or left-shifting the number n times.

Table 5.1: Arithmetic Elements required for certain transform lengths and radices

| N | FFT Radix | Real Multiplies | Real Additions |
|---|---|---|---|
| 256 | 2 | 4096 | 6144 |
| 256 | 4 | 3072 | 5632 |
| 256 | 16 | 2560 | 5696 |
| 512 | 2 | 9216 | 13,834 |
| 512 | 8 | 6144 | 12,672 |
| 4096 | 2 | 98,304 | 147,456 |
| 4096 | 4 | 73,728 | 135,168 |
| 4096 | 8 | 65,536 | 135,168 |
| 4096 | 16 | 61,440 | 136,704 |

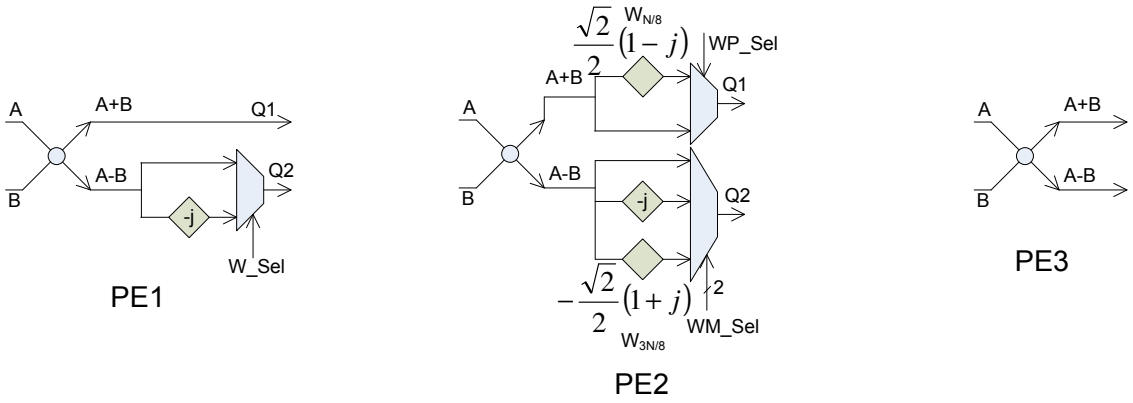


Figure 5.1: Processing Elements for the FFT Design Butterfly

This can be made use of to perform complex multiplication by $\sqrt{2}/2$ using only 12 additions as shown by Jia et.al [41]. For this constant multiplication, usage of 12 additions does not increase the error of the result. Thus, the Radix-2/4/8 Butterfly can be built without using any multipliers.

The complete Radix-2/4/8 Butterfly is as shown in Fig. 5.2.

Any FFT with number of points divisible by 8 will require a cascade of Radix-

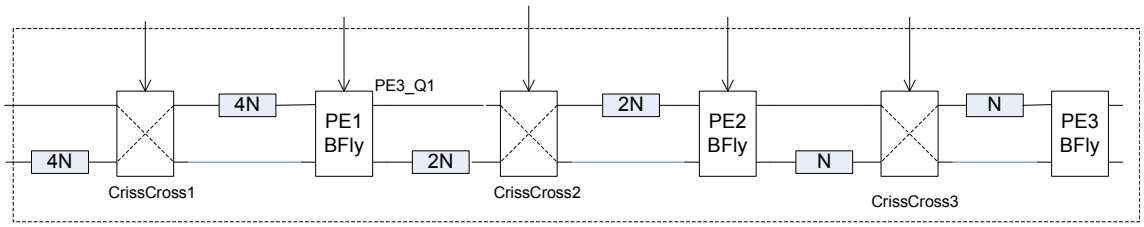


Figure 5.2: Radix 2/4/8 MDC FFT Butterfly

2/4/8 Butterflies. For other FFTs, Radix-2 or Radix-2/4 Butterflies can be padded with the cascade of Radix-2/4/8 Butterflies. For example, an 8192-pt FFT Processor can be built as shown in Fig.5.3.

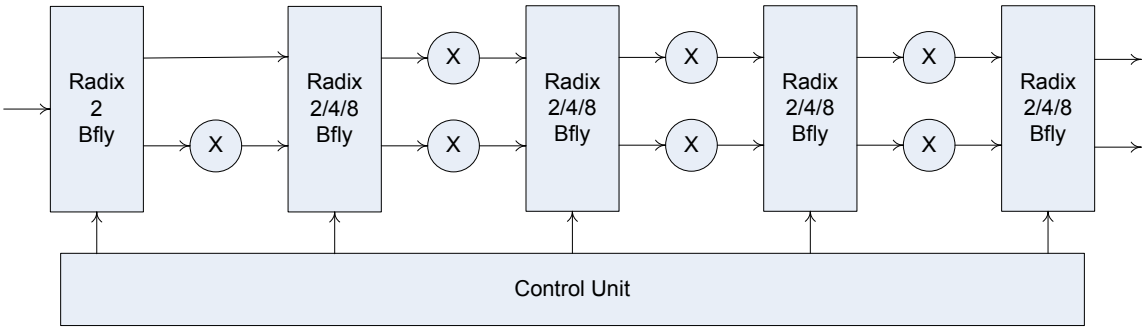


Figure 5.3: 8192pt Radix 2/4/8 MDC FFT Design

The control unit will control all the muxes inside the Butterflies. All multipliers will read twiddle factor data from individual ROMs which are not shown. The biggest twiddle ROM required for the design is a 4K words memory, where the word size will be 24 bits. It will also require 2 twiddle ROMs each of capacities 512, 64 and 8 words respectively. The FIFO memories required will be two each of 2K words, 1K words, 512 words and so on till 1 word of 24 bits.

## 5.2 Complex Multiplier Design

The complex multipliers are the biggest combinational blocks in the design and lie in the critical path. As the entire structure of the design is pipelined, it is functionally very easy to pipeline the complex multipliers and reduce the critical path delay.

We can use Eq.5.1 to calculate the complex multiplication [37].

$$(A+jB)*(C+jD) = [C*(A-B)+B*(C-D)]+j[D*(A+B)+B*(C-D)] \quad (5.1)$$

This requires 3 real multiplications and 5 real adders and can be built as shown in Fig.5.4.

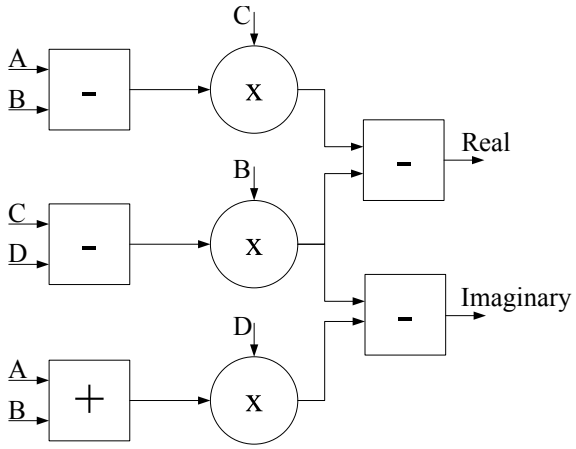


Figure 5.4: Complex Multiplier

The fixed-point VHDL package available from Doulos [6] is used to develop the RTL code for the complex multiplier. As the entire design has a flexible pipeline structure, it is easy to modify the number of pipeline stages if needed. This can be done by adding the appropriate number of registers in the datapath. We add registers

at the input and output of every complex multiplier in the design.

Care has to be taken for the twiddle factor table when register stages are added or deleted from the pipeline. As the data from the input samples reaches the complex multiplier one cycle later or earlier depending on whether a register stage was added or subtracted, the twiddle generator ROMs have to generate the twiddles one cycle earlier or later as the case maybe. This can be easily accomplished by barrel-shifting the contents of the Twiddle ROM as required.

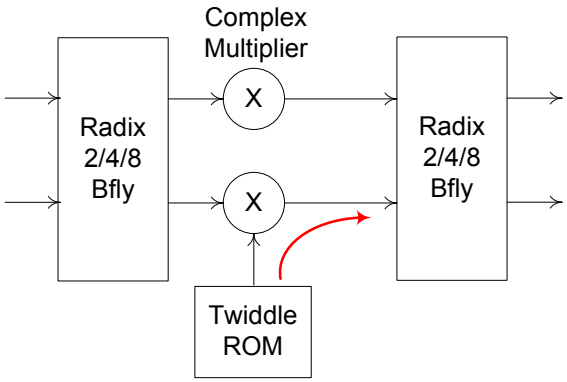


Figure 5.5: Critical Path of the Design

As shown in Fig.5.5, the critical path of the design lies through a read of the twiddle ROM and computation of the complex multiply operation. The critical timing can potentially be reduced (and critical path shifted to the FIFO logic) by adding further pipelining stages in the combinational logic of the complex multiplier. Again, any additional pipelining stages will cause a minor change in the twiddle ROM generation logic as explained above.

## 5.3 Partitioning

As shown in Section.3.4, the 3DIC flow starts with the synthesis and partitioning of the design. We use kmetis to partition most of the design between the three tiers, but perform the partitioning of the FIFOs manually like in Chapter 4. The FIFO memories are divided into three banks when used for the multi-tier designs and two banks when used for the conventional single-tier design. Thus, the RTL is specifically changed to suit the three-tier design structure.

After synthesis, the different memories on each tier are grouped together and extracted out of the netlist. The rest of the netlist is partitioned using kmetis and the memories are then added back to get the netlist for the entire design. The top module of this netlist will have nothing but three sub-modules which correspond to the design on the three tiers. These three pseudo-independent designs are then taken through the physical design flow as shown in the next section.

## 5.4 Physical Design of multi-tier FFT

This section describes the different steps taken during the physical design of the FFT design.

### 5.4.1 Floorplan of FFT

As shown in section 3.4.2, the input to the floorplanning step is the partitioned netlist. Since Cadence Encounter can only operate on single-tier designs, we need a

way to tie up the layouts of each tier into something legally cohesive.

For this design to work after fabrication, the MIT 3DIC technology and the standard cell library we designed imposes certain restrictions on the design. Some of the restrictions from the point of view of a physical designer are as follows:

1. All the three tiers should have the exact same dimensions. We take care of this restriction by using identical floorplan dimensions for each tier.

2. The Input-Output pads can only be present on TierC. For the designs in this report, we do not actually place I/O pads as that does not affect the performance characteristics significantly. Consistency is maintained by not placing pads on the single-tier designs as well. The original flow [38] manually put all the I/Os on TierC after the partitioning stage whereas we ignore this step.

3. The ITVs are placed such that the sub-parts of the same ITV should be placed at exactly the same location on different tiers. For example, for an VAC or VABC ITV, all the three sub-parts should lie at the exact same coordinates on the three tiers. As will be shown in later sections, scripts have been written to enforce this rule while placing the design.

We start with the floorplan for the design. The dimensions for the floorplan were chosen by starting with a density of 0.5 combined for the standard cells and the memories. TierB has the biggest area and the placement density was chosen for this tier. After the design was taken through the entire flow with this density, the next step was to see if a lower density was feasible. We finally reduced the density to 0.67

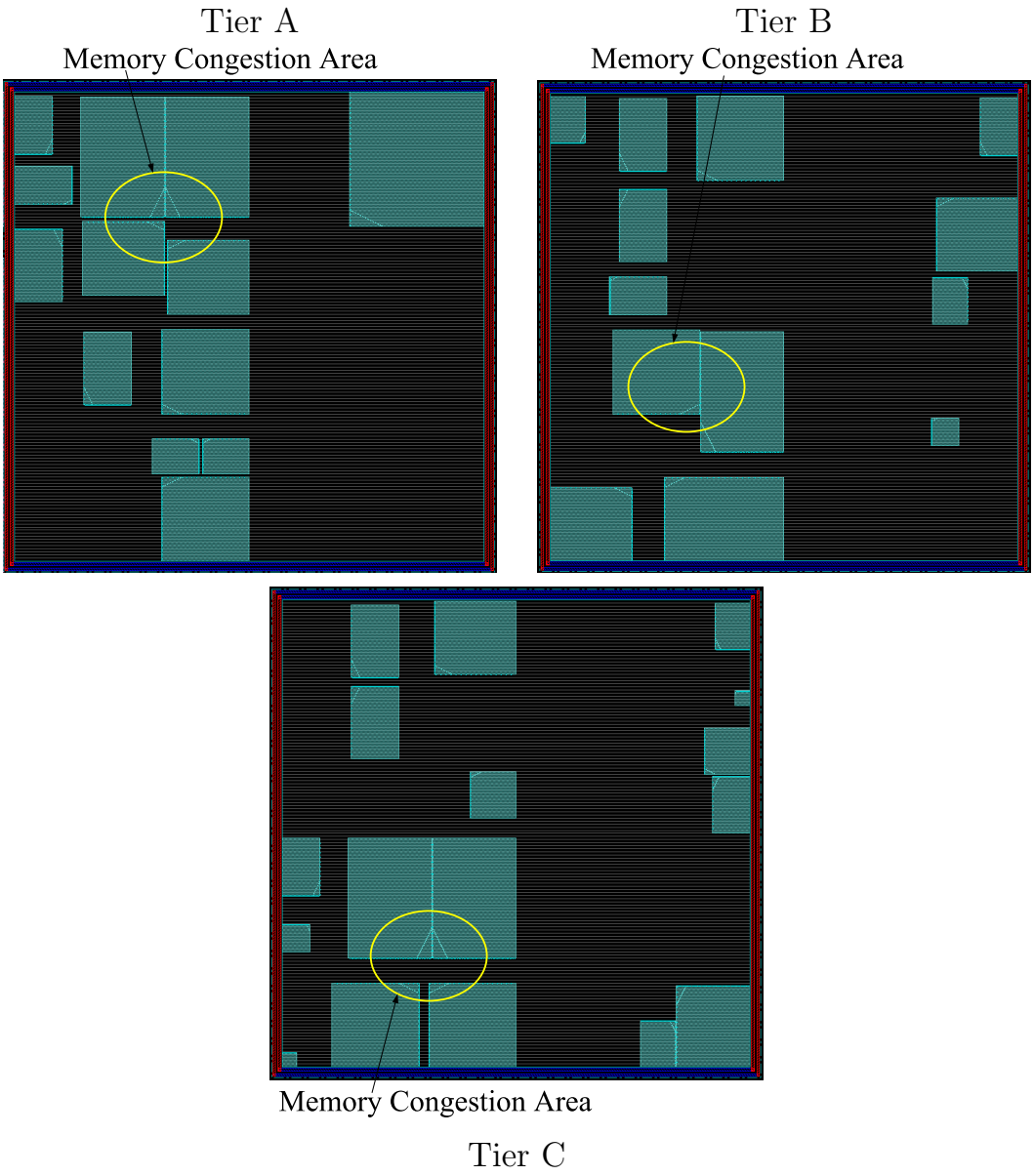


Figure 5.6: Initial floorplan with no constraints

for TierB and all the layout pictures will be shown for this density.

Our initial floorplan is the one generated without any constraints by Encounter. Figure 5.6 shows the initial floorplan for Tiers A,B and C respectively with a density

of 67%.

Each of these initial no-constraint floorplans from figure 5.6 has some common flaw in them. All of them have sections of floorplan where two or more memories are extremely close together. This is especially marked for each of the floorplans. On top of that, occasionally the sides of two memories, which almost touch each other, have pins on them, thus making the routability of such a design extremely difficult. The floorplanner brings components closer to reduce the average wirelength of the design, but it is bad from the router's point of view.

Also, this being a multi-tier design, the floorplans of each tier cannot be totally independent of each other. Since we have partitioned the FIFOs in the design to take advantage of the multi-tier feature, it is advisable that the sub-memories of each FIFO be at the same locations for each tier. Also, the twiddle ROMs in the design are not partitioned and hence can be put anywhere on its given tier, provided it does not overlap the location of a FIFO.

With these constraints in place, the floorplanning stage was accomplished using a trial-and-error method approach. For any incremental change (possibly improvement) in the floorplan, the design was routed to see if it was a feasible floorplan. Refinements were done till a routable design was achieved. Once a routable design was found, it was saved as a possible solution. The area of the design was reduced and the entire procedure was repeated to see if a more compact routable solution could be found.

We took advantage of the fact that the memories had pins on only two sides. That left two sides which could possibly touch the edges of the floorplan. Thus, preference

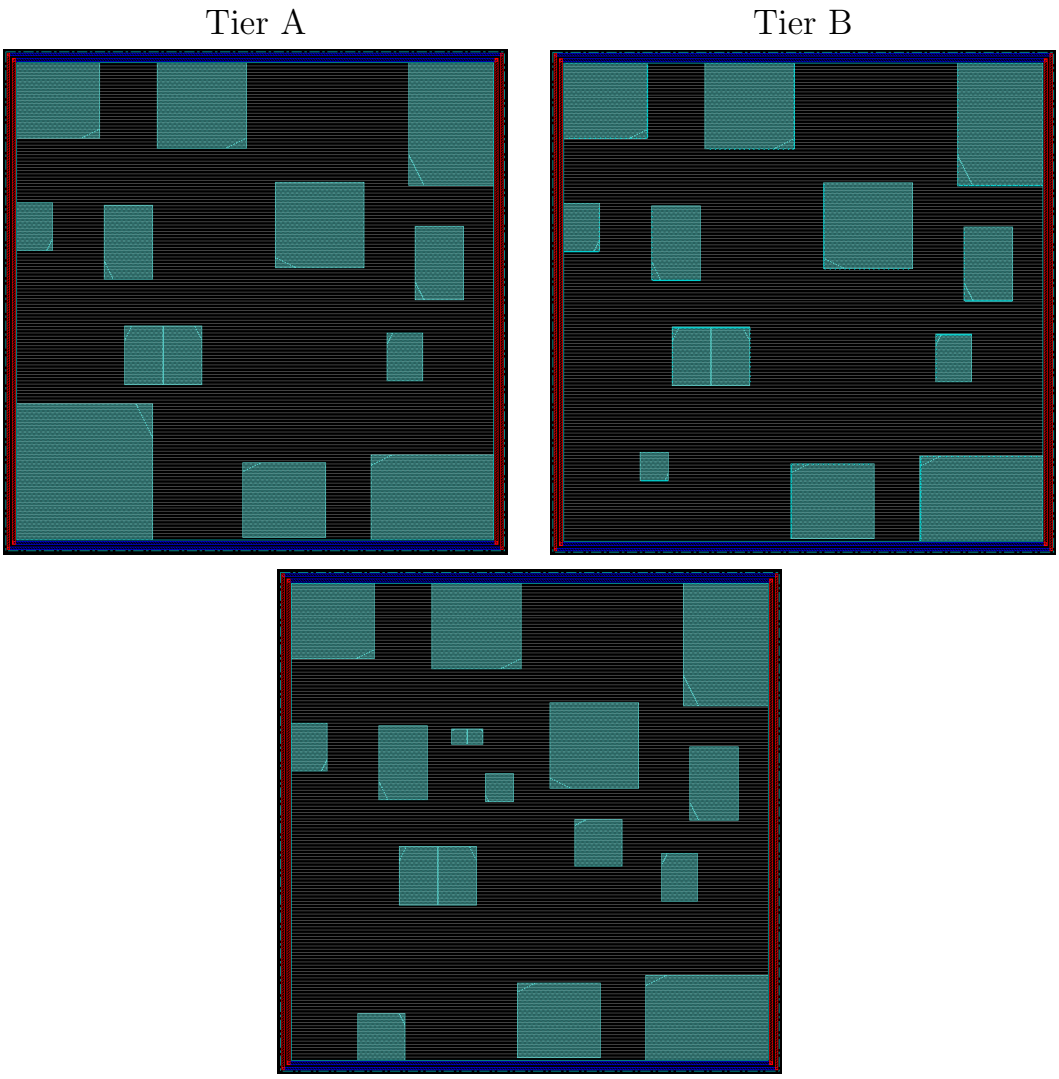


Figure 5.7: Floorplan

was given to place four memories at the four corners of any Tier. The ROMs in the design were not partitioned, whereas the RAMs were partitioned and hence had 3 sub-parts for each of the memory.

The biggest memory of all was the 4K ROM which was placed at the lower leftmost

corner of TierA. For all the RAM memories, they were first placed in TierB and a script was executed which copied their location on Tiers A and C for the other sub-parts of each memory. All the other ROMs were then placed on their respective tiers at places which were not blocked by previously placed memories. Figure 5.7 shows the final memory placement locations for Tiers A, B and C respectively.

### 5.4.2 Via Placement

After the memories are placed to their final locations, it is the turn of the Inter-tier vias to be placed. Again, the ITVs are to be placed such that the sub-parts of an ITV should be placed at the exact same location in the tiers which it connects. For this to happen, there should be no memory already placed at the location where a potential ITV sub-part wants to place itself. Thus, even the memories on TierA and TierC would act as obstructions for ITVs to be placed on TierB.

A script was executed which copied the memory locations of the ROMs on Tiers A and C and placed them as obstructions on TierB, as shown in Fig. 5.8. The red boxes represent the obstructions.

The original 3DIC flow [38] independently places the vias on each tier on a trial basis. Then, for each set of sub-parts of ITVs, it tries to place the entire set at the average location from these trial runs. This method is good when there are few or no memory obstructions in the design. With so many memories as present in the FFT design, it is likely that the average location for the set of ITVs would fall at a place where a memory is already placed. Extra code would be required to find a legitimate

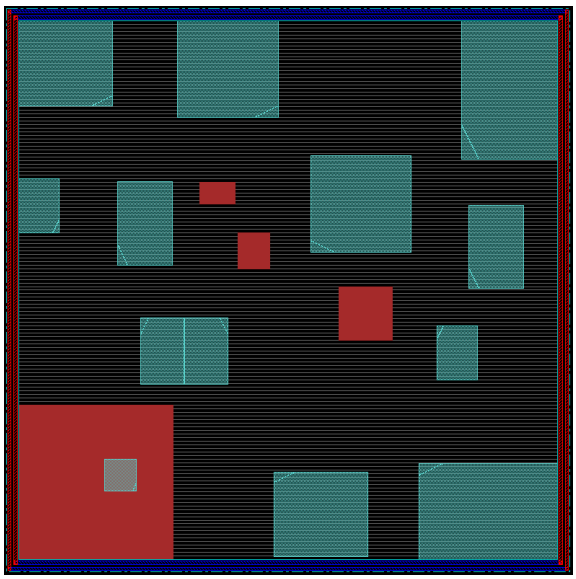

Figure 5.8: Tier B Floorplan with memory obstructions from Tiers A and C

place for each ITV.

We simplify this via placement process by noting that every ITV has at least one sub-part on TierB. Hence, we choose the trial via placement run on TierB as the final location for the ITVs. Since the obstructions from Tiers A and C are already copied onto TierB, we can be sure that this will generate a legal placement for each of the ITVs. The corresponding sub-parts for the ITVs on Tiers A and C are placed by copying their locations from the vias placed on TierB. The final via and memory placed floorplans for Tiers A, B and C are shown in Figure 5.9.

It is pertinent to note that this via-placement algorithm can be improved by splitting the trial run on TierB into three. For placing ITVs VAB, it is not needed to create obstructions due to memories on TierC. Similarly, for placing ITVs VBC, it is not needed to create obstructions due to memories on TierA. Thus, the trial run can be divided into 3 parts as follows:

1. In the first part, obstructions from memories on TierA are put on TierB and ITVs VAB would be placed.

2. In the second part, obstructions from memories on TierA will be removed, those on TierC will be added and ITVs VBC would be placed.

3. Lastly, all the obstructions as shown in Fig.5.8 would be added to TierB and the remaining ITVs viz.VAC and VABC would be placed.

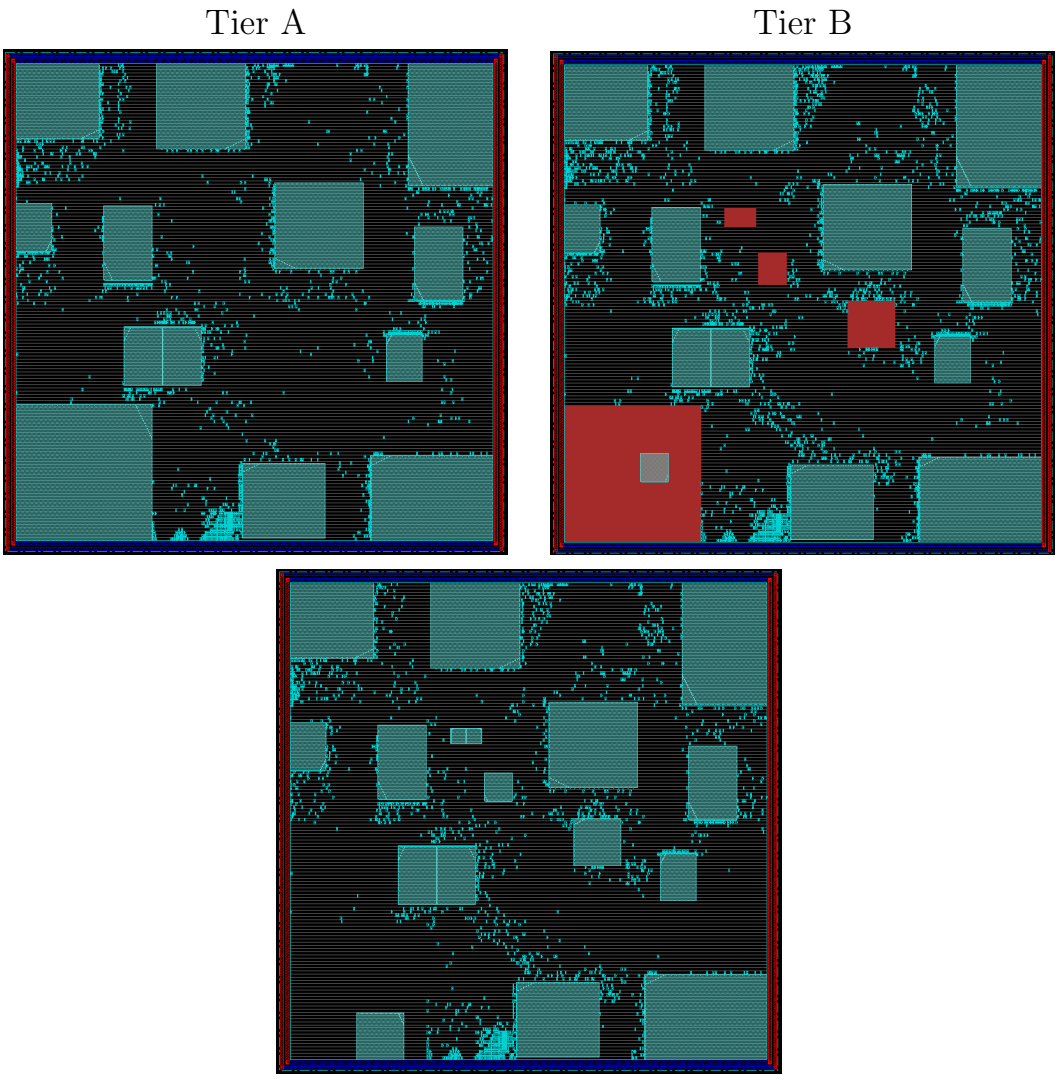


Figure 5.9: Inter-tier vias placed

### 5.4.3 Module Placement

After the vias are placed, the placer can be invoked on the entire design, one tier at a time. As the dependencies between the tiers due to memories and ITVs have already been taken care of, this placement run can be executed independently for

each tier. All the obstructions from TierB are removed before the placer is run. The Cadence Encounter tool "amoebaPlace" is used to place the tiers.

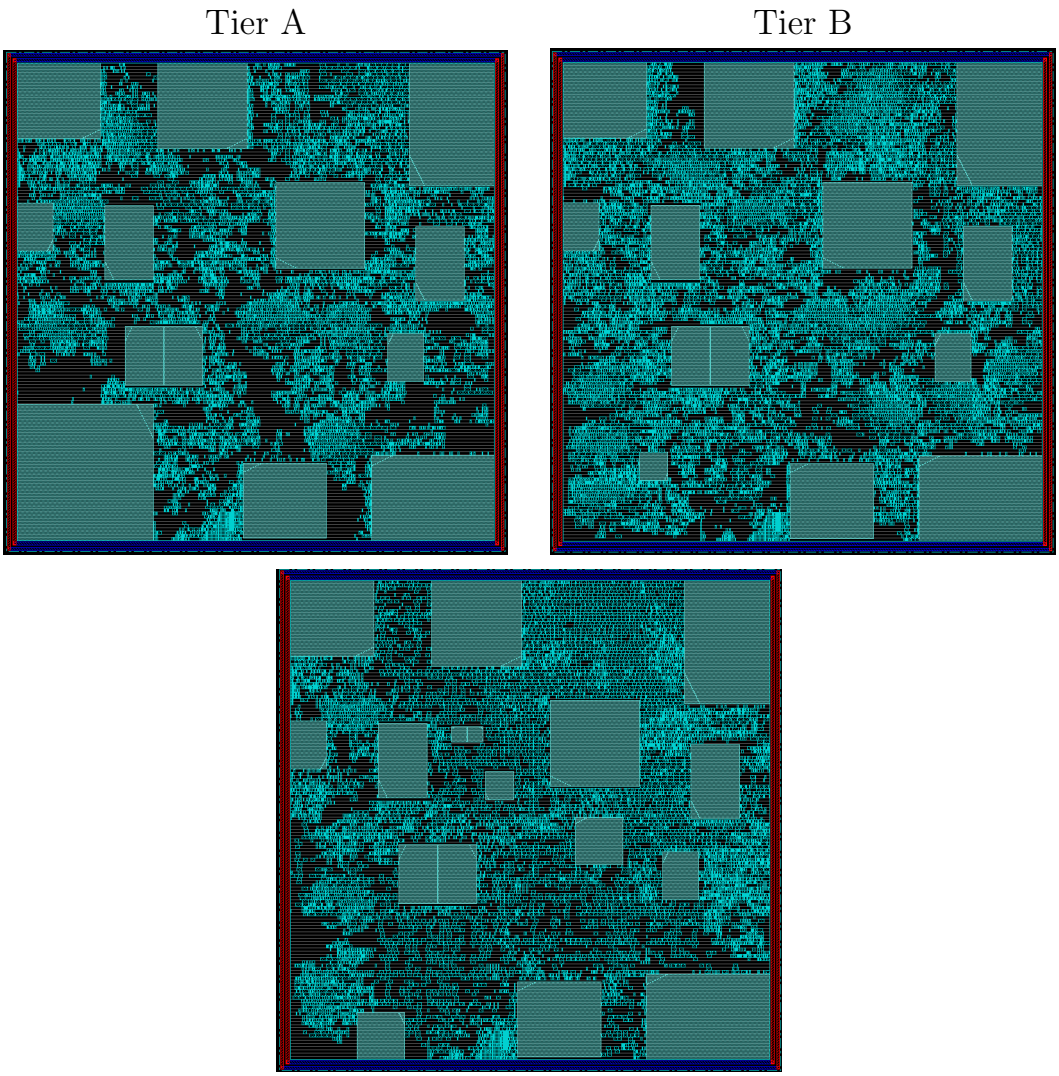


Figure 5.10: Final Placement

Figure 5.10 shows the final placed version of Tiers A, B and C. The overall density for TierA is 63%, for TierB is 67% and that for TierC is 66%.

### 5.4.4 Clock Tree Synthesis

The original 3DIC flow [38] independently places special clock vias VABC between tiers A, B and C to stitch the clocks on all tiers together. The sub-parts of the vias are aligned as usual to make sure that the insertion (or starting) points for the 3 clock networks is at the same location. We skip the alignment step but note that this should not cause any noticeable effect on the timing or power requirements of the design.

The CTS tree is restricted to have a maximum skew of 0.1ns between any two nodes in the tree. The reset trees are similarly routed in all the three tiers. Figure 5.11 shows the final placed versions of Tiers A, B and C with synthesized clock trees respectively.

### 5.4.5 Routing

After the insertion of clock and reset trees, the design is first trial routed and then warp-routed to get the final layout for the FFT.

Figure 5.12 shows the final routed versions of Tiers A, B and C respectively.

Parasitics are extracted from each tier to give three different SPEF files. Also, each of the three tiers will give a netlist which will be composed of the clock and reset tree buffers added as well. The three netlists along with the three SPEF files are then read into Synopsys Primetime and timing is reported. Similarly, they are read into Synopsys Primepower and power is reported. These performance characteristics of the design are reported in detail in chapter 6.

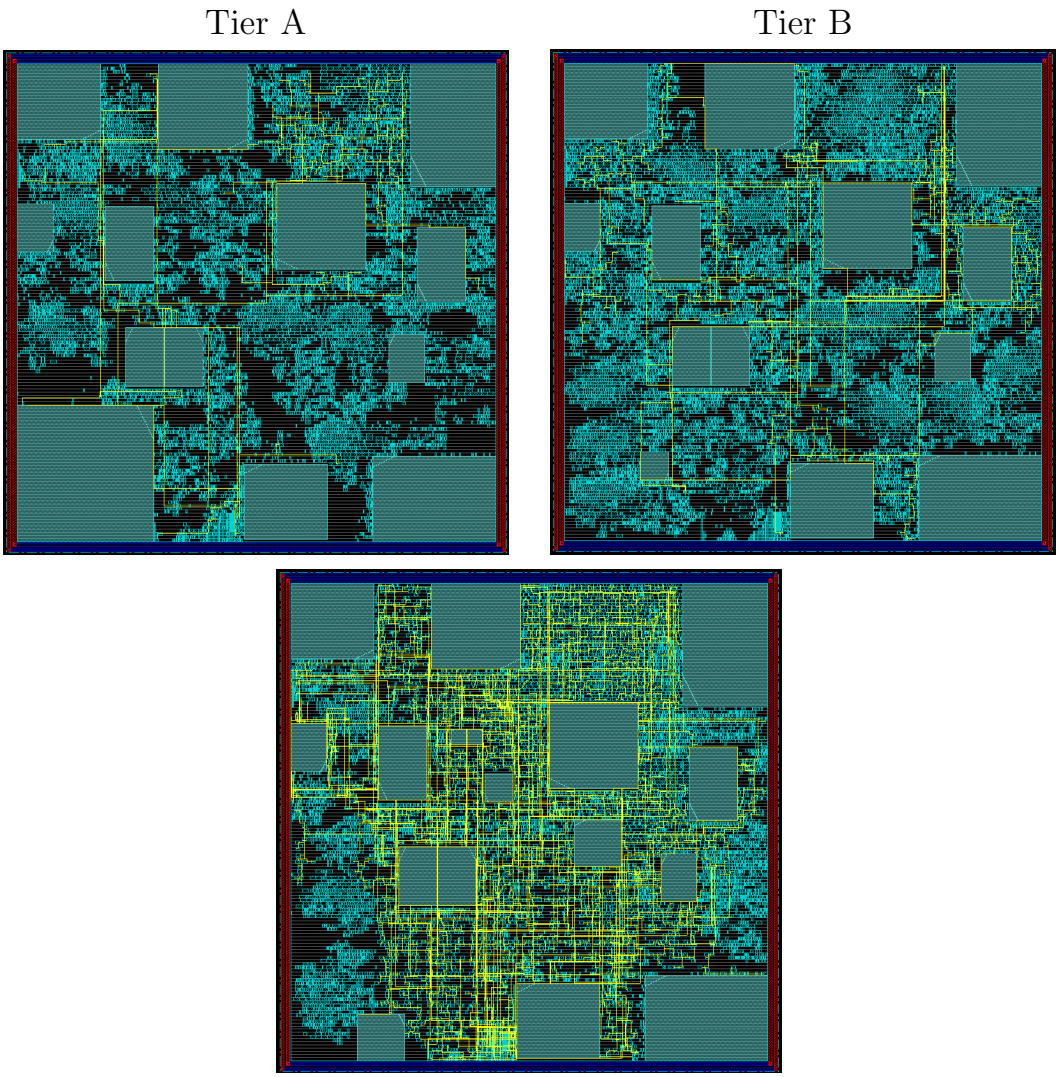


Figure 5.11: Final Synthesized Clock Tree

## 5.5 Physical Design of single-tier FFT

For the single-tier design, the delay buffers in the design were functionally split into 2 single-port memory banks each, as shown in chapter 4. Thus, each of the memories used to make up the delay buffer in the single-tier design are bigger and

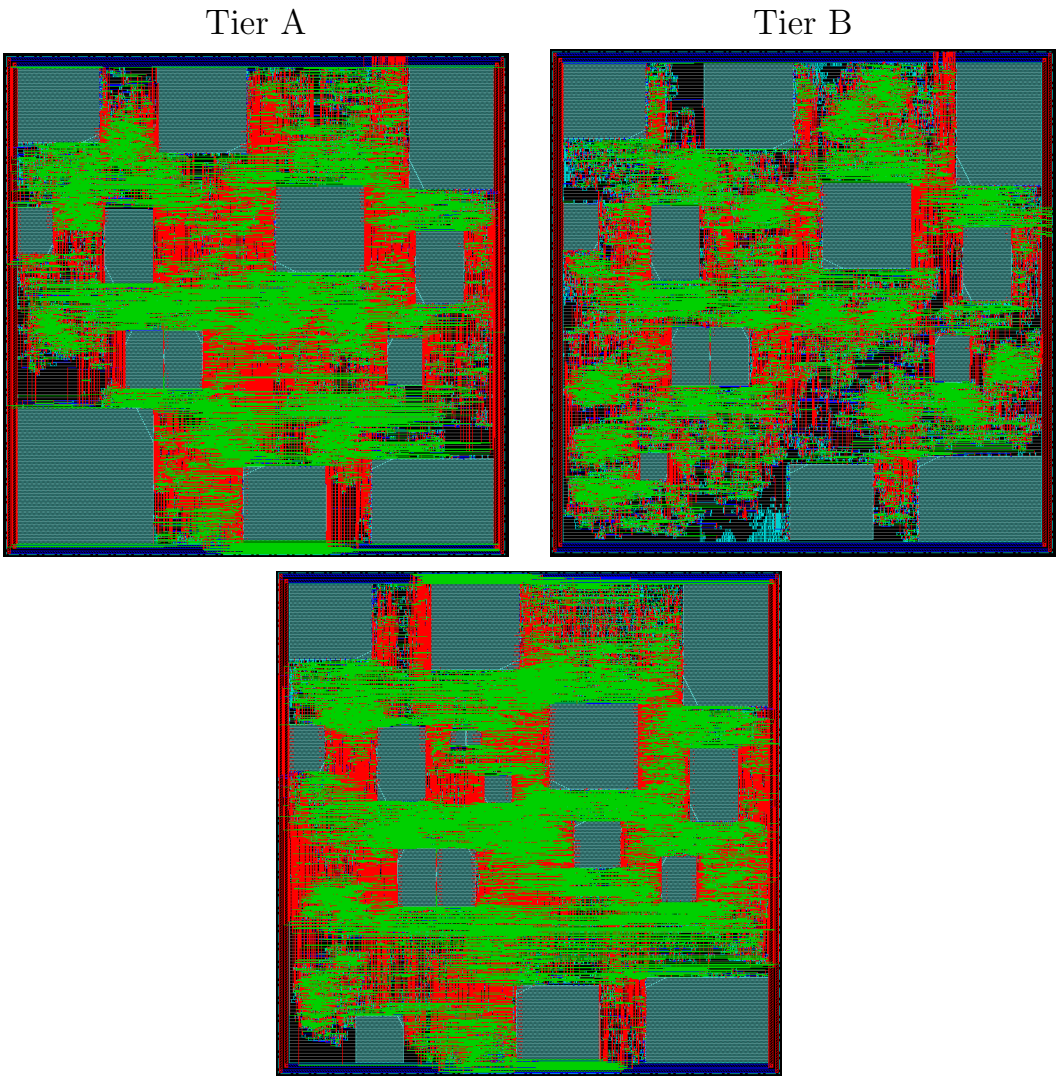


Figure 5.12: Final Routed Design

consequently slower than the ones used in the multi-tier design. The wire-lengths in the surrounding logic, however, are shorter. As described in chapter 4, the two-bank structure tends to be slightly faster overall than the three-bank structure.

The floorplan was made as close as possible to the multi-tier design. The difference

being that for every FIFO memory on a particular tier in the multi-tier design, there were 2 sub-banks of memories to be placed in the single-tier design.

Our initial floorplan is the one generated without any constraints by Encounter. Figures 5.13 shows the initial floorplan for the FFT design with a density of 67%.

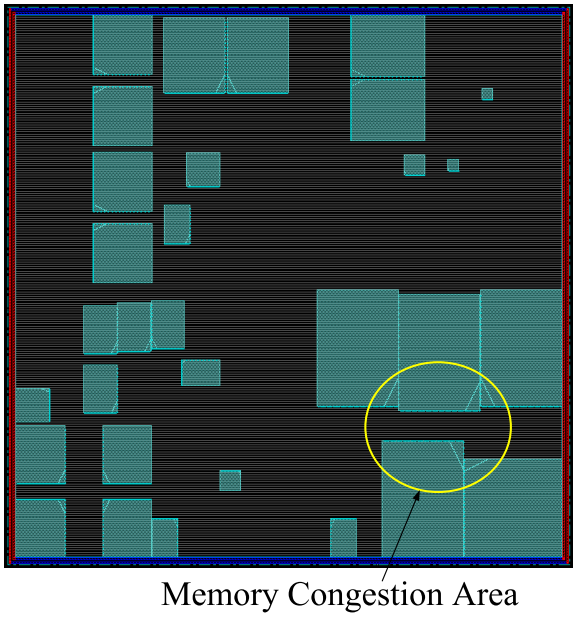


Figure 5.13: Single-tier FFT Floorplan with no constraints

For each of the FIFO memories, the 2 sub-banks were placed close to each other. This is to reduce the lengths of the wires connecting these two sub-banks and its controllers. The placer is expected to place the standard cells comprising its control logic close by to each FIFO.

Figure 5.14 shows the initial floorplan with memories fixed at their final locations. One pair of FIFO memory sub-banks are circled in yellow to highlight their proximity to each other. Similarly, other pairs of FIFO memories are clearly seen close to each

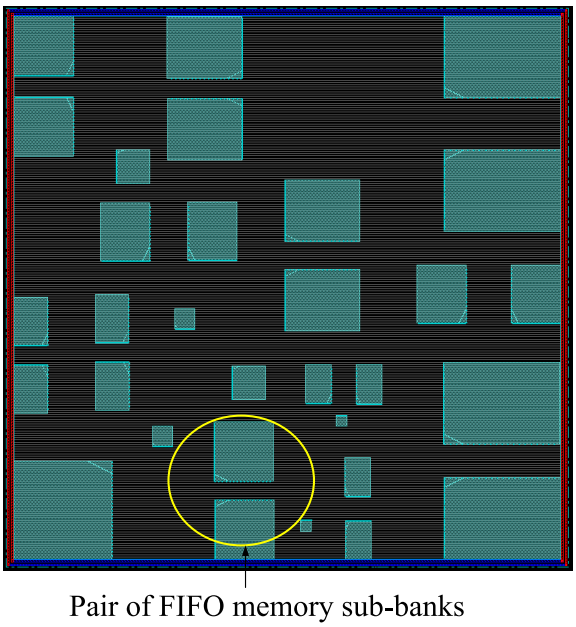


Figure 5.14: Single-tier FFT Initial Floorplan

other. The memories which do not have a identical memory placed close to them are the twiddle ROMs in the design.

After the floorplanning stage, the placer is run on all the modules. The ITV placement stage is not needed as there are no ITVs for the single-tier design. Figure 5.15 shows the design with all the standard cells placed in addition to the FIFO memories and twiddle ROMs.

The CTS stage follows the placement. Again, as in the multi-tier design, the CTS tree is restricted to have a maximum skew of 0.1ns between any two nodes in the tree. Figure 5.16 shows the design with the clock tree synthesized.

Finally, the design is trial and warp-routed to get the final layout for the single-tier FFT processor. Again, the performance characteristics of the single-tier design

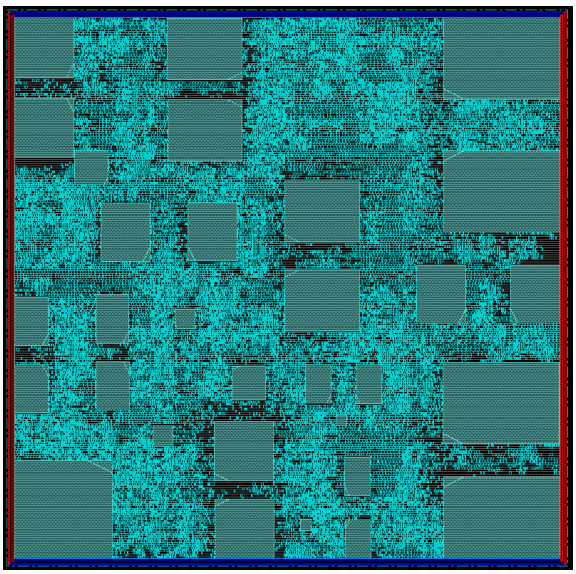

Figure 5.15: Single-tier FFT Placed Design

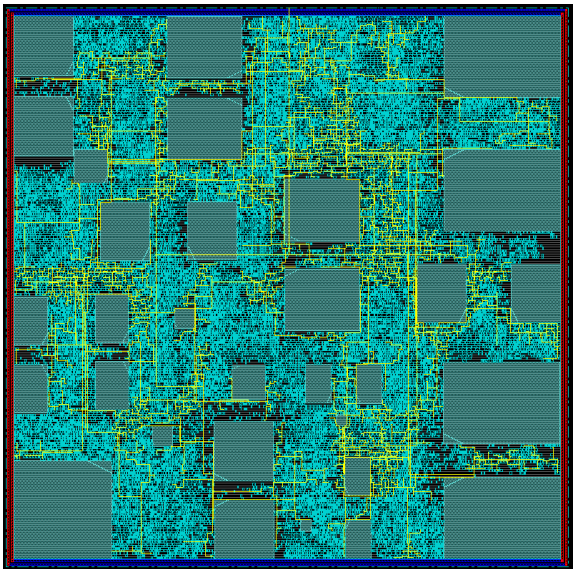

Figure 5.16: Single-tier FFT Placed Design with clock trees inserted

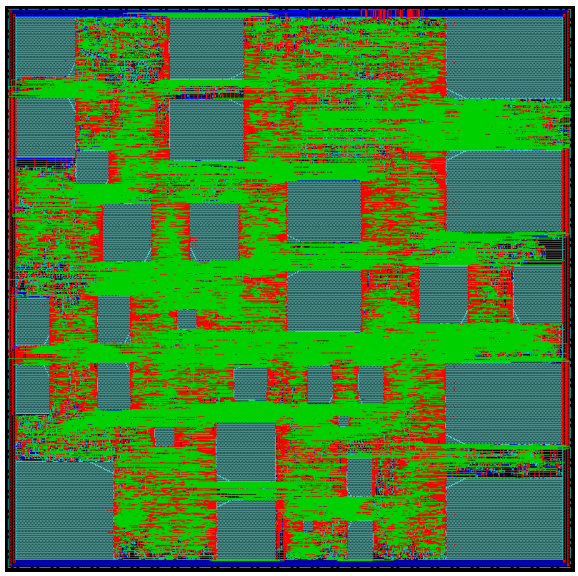

Figure 5.17: Single-tier FFT Final Routed Layout

are reported in detail in chapter 6. The performance of the single-tier and multi-tier design are also compared with three other contemporary 8192-point FFT processors.

# Chapter 6

# Results

This chapter will show the performance characteristics for the single-tier and multi-tier 8192 point FFT design which was described in chapter 5. The performance is then compared with three other 8192 point FFT designs published in related technical literature.

## 6.1 Performance Analysis

We start with the performance of the multi-tier design. Following the 3DIC flow described in chapter 3, we obtain 3 SPEF files depicting the parasitics for the three layers of the designed FFT processor. Each of the three tiers will also give a verilog netlist which will include the clock and reset buffer trees that were added during the CTS phase. The three netlists along with the three SPEF files are then read into Synopsys Primetime and the critical path timing for the unified design is reported.

This comes out to be 4.66 ns. The critical path delay is considered as the minimum clock period required so that the design does not fail due to setup time violations. Thus, the maximum clock frequency at which the design can be run is 214 MHz.

For the power analysis using Synopsys PrimePower, the design is assumed to be running at 214 MHz. The power dissipated by the numerous memories was not annotated in the standard cell library designed for the design. Instead the power numbers for the memories were manually added later. Without considering the memory, Synopsys PrimePower reported the power dissipated as 381 mW at 214 MHz. Using the clock period of 4.66 ns gives the energy used for one clock cycle as 1.78 nJ. Since the FFT takes 4096 cycles to calculate one set of FFTs, the execution time will be $19\mu s$.

We now consider the energy requirements for the memories. The FIFO memories are used as delay buffers and for every such delay buffer (made of 3 sub-banks of memories), we expect one read and one write to take place during every clock cycle. Thus, we assume that the energy used during one cycle is the addition of the read energy and write energy for one sub-bank which makes up the delay buffer. CACTI [68] reports the read and write energy for one access to the memory we are interested in. In the multi-tier design, the FIFO memory with a capacity of 2KB words is designed using 3 sub-memories of 700 words. In this case, the energy requirement of the 2KB words FIFO per cycle is assumed to be the sum of the read energy and the write energy for the 700 words memories. In this manner, the energies for all the memories are added up and this figure comes out to 0.105 nJ. We observe that for a $0.18\mu m$ process commercially designed memory, the leakage power would be extremely small

and so would the dynamic energy. For this reason, the total energy requirement of all the memories combined is very small compared to the energy required by the rest of the combinational logic.

Thus, the total energy requirement for the multi-tier FFT is the addition of the energy requirement for the memories (0.105 nJ) to that of the rest of the design (1.78 nJ). This is 1.88 nJ during one cycle and 7676 nJ for 4096 cycles which is the time taken for one set of FFT.

For the single-tier design, getting performance characteristics is easier as any merging of tier-specific parasitics is not needed. After extraction of the parasitics from the final routed design, timing analysis was done using Synopsys Primetime and the critical path delay was reported to be 5.98ns. Thus, the maximum clock frequency at which the design can be run is 167 MHz. Synopsys PrimePower reports the power dissipated at 167 MHz to be 367mW without the memories. Thus, the energy used during one clock cycle comes out to 2.19nJ.

The energy requirement of the memories is calculated similar to the multi-tier design and it is 0.114 nJ. Thus, the total energy requirement for the single-tier FFT becomes 2.31 nJ during one cycle and 9394 nJ for 4096 cycles (time taken for one set of FFT).

## 6.2 Comparison

The performance characteristics of the proposed Radix 2/4/8 MDC designs are added to Table 2.2 to compare the parameters with the previously mentioned designs.

Table 6.1 shows this comparison.

Table 6.1: Comparison with other 8192-pt FFT Processors

| | Jia [41] | Bidet [13] | Lin [53] | Proposed 2D | Proposed 3D |
|---|---|---|---|---|---|
| Word length (bits) | 12 | 12 | 11 | 12 | 12 |
| Radix | 2/4/8 | 4 | 2/4/8 | 2/4/8 | 2/4/8 |
| Process, $\mu m$ | 0.6 | 0.5 | 0.18 | 0.18 | 0.18 |
| $K$ | 3.33 | 2.77 | 1 | 1 | 1 |
| Voltage, V | 3.3 | 3.3 | 1.8 | 1.5 | 1.5 |
| Power, mW $\downarrow$ | 650 | 600 | 25.2 | 385 | 404 |
| Norm. Power, mW $\downarrow$ | 58.5 | 77.7 | 25.2 | 385 | 404 |
| Clock Freq, MHz $\uparrow$ | 20 | 20 | 20 | 167 | 214 |
| Norm. Clock Freq, MHz $\uparrow$ | 66.6 | 55.5 | 20 | 167 | 214 |
| Execution Time, $\mu s$ $\downarrow$ | 400 | 400 | 717 | 24.4 | 19 |
| Norm. Exec Time $\mu s$ $\downarrow$ | 120 | 144 | 717 | 24.4 | 19 |
| Energy, $\mu J$ $\downarrow$ | 260 | 240 | 18.07 | 9.3 | 7.6 |
| Norm. Energy $\mu J$ $\downarrow$ | 7 | 11.2 | 18.07 | 9.3 | 7.6 |
| Area, $mm^2$ $\downarrow$ | 107 | 100 | 4.84 | 8.17 | 10.6 |
| Norm. Area, $mm^2$ $\downarrow$ | 9.63 | 12.96 | 4.84 | 8.17 | 10.6 |
| FFT/energy, $mJ^{-1}$ $\uparrow$ | 12.54 | 11.57 | 55.32 | 105.87 | 129.65 |
| Norm. F/E, $mJ^{-1}$ $\uparrow$ | 139.3 | 89.2 | 55.32 | 105.87 | 129.65 |
| Energy * Time ($\mu J$*ns) $\downarrow$ | 104000 | 96000 | 12955 | 229.2 | 145.8 |
| Norm. E * T ($\mu J$*ns) $\downarrow$ | 842.4 | 1612.4 | 12955 | 229.2 | 145.8 |

As can be seen from this table, when comparing with normalized execution time, the 3D design ($19\mu s$) is 84% better than the best execution time ($120\mu s$) among the previous three designs. It is slightly worse by 8% when comparing the normalized energy ($7.6\mu J$ v/s $7\mu J$), but is much better (82%) in terms of energy-time product (145.8 v/s 842.4). The 3D design is 93% worse when comparing its power (404 mW) with

the lowest power design (25.2 mW). But since our focus is on high-speed of operation, that is understandable.

Just comparing the two proposed single-tier and multi-tier designs shows the advantage of 3DIC technology for pipeline FFTs. The multi-tier design is 22% faster (19 v/s 24.4), and 18% more energy efficient (7.6 v/s 9.3) than the single-tier design. In terms of total silicon area though, the single-tier design is 18% better (8.1 v/s 10.6) than the multi-tier design.

## 6.3 Conclusion

This work demonstrated an EDA flow to design memory-based circuits using multi-tier 3DIC technology. The major enhancement to an existing 3DIC flow [38] was the handling of multiple inter-dependent memories in different tiers. Since placement of the memories is a task which in general cannot be performed in an optimum way using automatic algorithms, some human intervention is required to perform this task iteratively on a trial and error basis.

This work also proposes a technique to build FIFO Buffer memories by leveraging 3DIC technology to reduce memory access times. A novel approach to designing the address pointers for the FIFO memories is shown. The utility of such FIFO structures is demonstrated by using them in the design of a FFT processor.

The radix 2/4/8 multi-path delay commutator architecture is used for the first time to design a pipeline FFT processor. The difference in performance between some previously published 8192-point FFT designs and the proposed single-tier and multi-

tier circuits are highlighted. After accounting for technology differences, the proposed 3D design is 84% faster than the previous fastest design and 82% better in terms of energy-delay product, but uses 8% more energy and 93% more power compared to the best designs in those categories.

The increase in speed for the proposed designs can be mainly attributed to the use of the MDC architecture which generates two output sample points per cycle compared to the SDF architecture which generates one output sample point per cycle. Although the MDC architecture provides double the throughput of the SDF one, it comes with some disadvantages. Compared with the radix 2/4/8 SDF designs, the proposed designs use more number of complex multipliers (7 v/s 4) and require more commutators. The proposed designs also require double the amount of input and output external memory bandwidth. The increased hardware leads to more area and power. Unlike the SDF designs, the MDC ones can be used with 100% utilization only when two independent sets of FFTs are processed simultaneously and external memory is used to convert each input set into two parallel data streams to be fed to our design, one after another. When the pipeline is thus filled with 100% utilization, the 2X throughput can be achieved.

Comparing the two proposed designs, we observe that the multi-tier design is 22% faster, and 18% more energy efficient than the single-tier design. This is due to reduction in sizes of memories and compaction of design leading to smaller average wirelengths. With proper designing, we can thus expect multi-tier pipeline FFT processors to work faster and utilize lesser energy when compared to comparable

single-tier designs.

## 6.4 Future considerations

We observe that the performance of such designs using a lot of different memories depends heavily on how the memories are placed. An optimal floorplan is difficult to design and even small changes to the floorplan could potentially make the design unroutable. Our case study showed a particular way of designing the pipeline FFT. Various other configurations of the memory structures could be tried out to give vastly different floorplans and consequently changed performance. Examples of changes would be dividing the FIFO memories into two tiers instead of three, two banks per tier instead of one or various other combinations.

To explore and compare the performance of bigger and more complex circuits, robust true 3D-aware physical design tools need to be designed. These tools need to have a good understanding of thermal analysis of the 3D circuits so that they could be reliably manufactured. This would lead to proper methodologies for addition of thermal vias throughout the design and around any memories, leading to proper dissipation of heat.

FFT architectures which require more area and memory bandwidth could potentially be developed using 3DIC technology, if the advantages justify the added cost. The techniques shown for designing the FIFO and FFT circuits could conceivably be used in other signal processing applications where memory access time is a bottleneck.

# Bibliography

[1] Cadence Encounter Infosheet. http://www.cadence.com/products/digital_ic/first_encounter/index.aspx.

[2] Pathfinder-2 ASIC FFT processor by DRS Technology. http://www.drs.com/products/index.cfm?gID=20&productID=450.

[3] Synopsys Design Compiler Infosheet. http://www.synopsys.com/products/logic/design_compiler.html.

[4] Synopsys PrimePower. http://www.synopsys.com/products/solutions/galaxy/power/power.html.

[5] Synopsys PrimeTime. http://www.synopsys.com/products/analysis/primetime_ds.html.

[6] VHDL Fixed Point Arithmatic Package. http://www.doulos.com/knowhow/vhdl_designers_guide/models/fp_arith.

[7] C. Ababei, Y. Feng, B. Goplenand, H. Mogal, T. Zhang, K. Bazargan, and S. Sap-

atnekar. Placement and Routing in 3D Integrated Circuits. *IEEE Transactions on Design and Test of Computers*, pages 520–531, Nov. 2005.

[8] Bevan Baas. *An approach to Low Power High performance Fast Fourier Transform Processor Design*. PhD thesis, Stanford University, 1999.

[9] K. Balakrishnan, V. Nanda, S. Easwar, and S. K. Lim. Wire congestion and thermal aware 3D global placement. In *IEEE/ACM Asia South Pacific Design Automation Conference*, pages 1131–1134, 2005.

[10] K. Banerjee, S.J. Souri, P. Kapur, and K.C. Saraswat. 3DICs: a novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. *Proceedings of the IEEE*, 89(5):602–633, 2001.

[11] Glenn D. Bergland. A fast fourier transform algorithm using base 8 iterations. *Mathematics of Computation*, 22(2):275–279, April 1968.

[12] R. Bhatia, M. Furuta, and J. Ponce. A quasi radix-16 VLSI processor. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 1085–1088, 1991.

[13] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn. A fast single-chip implementation of 8192 complex point FFT. *IEEE Journal of Solid-State Circuits*, 30(3):300–305, 1995.

[14] B. Black, D.W. Nelson, C. Webb, and N. Samra. 3D processing technology and its impact on IA32 microprocessors. In *Proceedings of the IEEE International*

*Conference on Computer Design: VLSI in Computers and Processors*, pages 316–318, 2004.

[15] D.R. Bungard, L. Lau, and T.L. Rorabaugh. Programmable FFT processors for wide-bandwidth HF spread-spectrum communications and radar signal processing. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 1357–1359, 1989.

[16] J. Burns, L. McIlrath, J. Hopwood, C. Keast, D.P. Vu, K. Warner, and P. Wyatt. An SOI-based three-dimensional integrated circuit technology. In *IEEE SOI Conference*, pages 20–21, October 2–5, 2000.

[17] C. Burrus and P. Eschenbacher. An in-place, in-order prime factor FFT algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(4):806–817, 1981.

[18] C.S. Burrus and T.W. Parks. *DFT/FFT and Convolution Algorithms*. John Wiley & Sons, 1985.

[19] L.W. Chan and M.Y. Chen. A new systolic array for discrete fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(10):1665–1666, 1988.

[20] Sau-Gee Chen and Chao-Kai Chang. FFT Design for Broadband Communication. http://larc.ee.nthu.edu.tw/dtc/doc/0503.pdf.

[21] J. Cong, W. Jie, and Z. Yan. A thermal-driven floorplanning algorithm for

3D ICs. In *IEEE International Conference on Computer Aided Design*, pages 306–313, Nov. 2004.

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

[23] S. Das, A. Chandrakasan, and R. Reif. Design tools for 3D integrated circuits. In *IEEE Proceedings of the ASP-DAC*, pages 53–56, Jan. 2003.

[24] W.R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A.M. Sule, M. Steer, and P.D. Franzon. Demystifying 3D ICs: the pros and cons of going vertical. *IEEE Design & Test of Computers*, 22(6):498–510, 2005.

[25] Dean M. Drako and Hsin-Tung A. Yu. Apparatus for alternatively accessing single port random access memories to implement dual port, Dec 1994. U.S. Patent Number: 5371877.

[26] R.J. Drost et al. Proximity communication. *IEEE Journal of Solid State Circuits*, 39(9):1529–1535, Sept. 2004.

[27] P. Duhamel and H. Hollmann. Split-Radix FFT Algorithm. *Electronic Letters*, 20(1):14–16, Jan 1984.

[28] P. Duhamel and M. Vetterli. Fast fourier transforms: a tutorial review and a state of the art. *Signal Processing*, 19(4):259–299, April 1990.

[29] A.R. Feldman and T. Van Duzer. Hybrid Josephson-CMOS FIFO. *IEEE Transactions on Applied Superconductivity*, 5:2648–2651, June 1995.

[30] L.R. Fenstermaker and K.J. O'Conner. A low-power generator-based FIFO using ring pointers and current-mode sensing. In *International Solid-State Circuits Conference*, pages 242–243, February 24–26, 1993.

[31] D. Garrett and M. Stan. Power reduction techniques for a spread spectrum based correlator. In *International Symposium on Low Power Electronics and Design*, pages 225–230, August 18–20, 1997.

[32] H. L. Groginsky and G.A. Works. A pipeline fast fourier transform. *IEEE Transactions on Computers*, 19(11):1015–1019, 1970.

[33] M. Hashimoto, M. Nomura, K. Sasaki, et al. A 20-ns 256K X 4 FIFO memory. *IEEE Journal of Solid-State Circuits*, 23(2):490–499, April 1988.

[34] Shouseng He and Mats Torkelson. A new approach to pipeline FFT processor. In *IEEE Computer Society International Parallel Processing Symposium*, pages 766–770, Apr. 1996.

[35] Shouseng He. and Mats Torkelson. Designing pipeline FFT processor for OFDM demodulation. In *International Symposium on Signals, Systems and Electronics*, pages 257–262, 1998.

[36] Shousheng He and M. Torkelson. Design and Implementation of a 1024-point Pipeline FFT Processor. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 131–134, May 11–14, 1998.

[37] Y.H. Hu. The quantization effects of the CORDIC algorithm. *IEEE Transactions on Signal Processing*, 40(4):834–844, 1992.

[38] Hao Hua. *Design and Verification Methodology for Complex Three-Dimensional Digital Integrated Circuit*. PhD thesis, North Carolina State University, 2006.

[39] Hao Hua, C. Mineo, K. Schoenfliess, A. Sule, S. Melamed, and W.R. Davis. Performance Trend in Three-dimensional Integrated Circuits. In *Interconnect Technology Conference, 2006 International*, pages 45–47, June 2006.

[40] Hao Hua, Chris Mineo, K. Schoenfliess, Samson Melamed, Ambarish Sule, Ravi Jenkal, and W.Rhett Davis. Exploring compromises among timing, power and temperature in three-dimensional integrated circuits. In *Proceedings of Design Automation Conference*, 2006.

[41] Lihong Jia, Yonghong Gao, Jouni Isoaho, and Hannu Tenhunen. A new VLSI-oriented FFT algorithm and implementation. In *Proceedings of the Eleventh Annual IEEE International ASIC Conference*, pages 337–341, Rochester, NY, 1998.

[42] V.N. Johnson, J. Jozwiak, and A. Moll. Through wafer interconnects on active pmos devices. In *IEEE Workshop on Microelectronics and Electron Devices*, pages 82–84, 2004.

[43] Douglas L. Jones. Decimation-in-Frequency Radix-2 FFT. http://cnx.org/content/m12018/latest.

[44] K. Kanda et al. 1.27Gb/s/pin 3mW/pin Wireless Superconnect (WSC) Interface Scheme. In *ISSCC*, pages 186–187, 2003.

[45] N. Kanopoulos and J. Hallenbeck. A first-in, first-out memory for signal processing applications. *IEEE Transactions on Circuits and Systems*, 33(5):556–558, May 1986.

[46] N. Kanopoulos and J. Hallenbeck. A first-in, first-out memory for signal processing applications. *IEEE Transactions on Circuits and Systems*, 33(5):556–558, 1986.

[47] G. Karypis and V. Kumar. The metis serial graph partitioning tool. http://www-users.cs.umn.edu/~karypis/metis.

[48] H. Kuriyama, T. Hirose, S. Murakami, T. Wada, K. Fujita, Y. Nishimura, and K. Anami. An 8 ns 4 Mb serial access memory. In *IEEE Journal of Solid-State Circuits*, volume 26, pages 502–506, April 1991.

[49] R. M. Lea et al. A 3-d stacked chip packaging solution for miniaturized massively parallel processing. *IEEE Transactions on Advanced Packaging*, 22:424–432, Aug. 1999.

[50] Moon-Key Lee, Kyung-Wook Shin, and Jang-Kyu Lee. A VLSI array processor for 16-point FFT. *IEEE Journal of Solid-State Circuits*, 26(9):1286–1292, 1991.

[51] Weidong Li and L. Wanhammar. A pipeline FFT processor. In *IEEE Workshop on Signal Processing Systems*, pages 654–662, Oct. 1999.

[52] Y. T. Lin, P. Y. Tsai, and T. D. Chiueh. Low-power variable-length fast Fourier transform processor. In *IEEE Proceedings in Computers and Digital Techniques*, pages 499–506, Jul. 2005.

[53] Yu-Wei Lin, Hsuan-Yu Liu, and Chen-Yi Lee. A dynamic scaling FFT processor for DVB-t applications. *IEEE Journal of Solid-State Circuits*, 39(11):2005–2013, 2004.

[54] Chih-Wei Liu. Introduction to FFT Processors. http://twins.ee.nctu.edu.tw/courses/vsp_04/handout/FFTProcessors.pdf.

[55] Gian Luca Loi, Banit Agrawal, et al. A thermally aware performance analysis of vertically integrated 3D processor-memory hierarchy. In *Proceedings of Design Automation Conference*, 2006.

[56] James Jian-Qiang Lu, Ken Rose, and Susan Vitkavage. 3D Integration: Why, What, Who, When? In *Future Fab International*, volume 23, Jul. 2007.

[57] Pronita Mehrotra. *Memory Intensive Architectures for DSP and Data Communication*. PhD thesis, North Carolina State University, 2002.

[58] S. Mick, J. Wilson, and P. Franzon. 4 gbps high-density ac coupled interconnection. In *IEEE Custom Integrated Circuits Conference*, pages 133–140, 2002.

[59] G. O'Leary. Nonrecursive digital filtering using cascade fast fourier transformers. *IEEE Transactions on Audio and Electroacoustics*, 18(2):177–183, 1970.

[60] Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing*. Prentice Hall, 1975.

[61] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits*. Prentice-Hall Publication, 2002.

[62] N. Shibata, M. Watanabe, and Y. Tanabe. A current-sensed high-speed and low-power first-in-first-out memory using a wordline/bitline-swapped dual-port SRAM cell. *IEEE Journal of Solid-State Circuits*, 37(6):735–750, June 2002.

[63] R. Singleton. An algorithm for computing the mixed radix fast fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 17(2):93–103, 1969.

[64] K. Siozios, K.Sotiriadis, V.F.Pavlidis, and D.Soudris. Exploring alternative 3D FPGA architectures: Design methodology and cad tool support. In *International Conference on Field Programmable Logic and Applications*, pages 652–655, Aug. 2007.

[65] H. Sorensen, D. Jones, M. Heideman, and C. Burrus. Real-valued fast fourier transform algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(6):849–863, 1987.

[66] Ambarish Mukund Sule and W. Rhett Davis. Designing FIFO buffers using 3DIC technology. In *24th International VLSI/ULSI Multilevel Interconnection Conference*, pages 267–272, Sept. 2007.

[67] V. Suntharalingam, R. Berger, et al. Megapixel CMOS image sensor fabricated

in three-dimensional integrated circuit technology. In *IEEE International Solid-State Circuits Conference. Digest of Technical Papers.*, pages 356–357, 2005.

[68] David Tarjan, Shyamkumar Thoziyoor, and Norman P. Jouppi. CACTI 4.0, June 2006. http://www.hpl.hp.com/techreports/2006/HPL-2006-86.pdf.

[69] H. Wang and P.C. Liu. Double-edge-triggered address pointer for low-power high-speed FIFO memories. *Electronics Letters*, 33(5):387–389, February 27, 1997.

[70] Neil H.E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI design : a systems perspective*. Addison-Wesley Pub. Co., 1993.

[71] S. Winograd. A new method for computing DFT. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 366–368, 1977.

[72] Gin-Der Wu and Ying Lei. Low Power Pipelined Radix-2 FFT Processor for Speech Recognition. In *IEEE/SMC International Conference on System of Systems Engineering*, April 24–26, 2006.

[73] Wen-Chang Yeh and Chein-Wei Jen. High-speed and low-power split-radix FFT. *IEEE Transactions on Signal Processing*, 51(3):864–874, Mar. 2003.

[74] Jung yeol Oh and Myoung seob Lim. Fast fourier transform processor based on low power and area efficient algorithm. In *Asia Pacific Conference on Advanced System Integrated Circuits*, pages 198–201, 2004.

[75] A. Zeng, J. Lu, K. Rose, and R.J. Gutmann. First-order performance prediction of cache memory with wafer-level 3D integration. *IEEE Design and Test of Computers*, 22(6):548–555, November/December 2005.

[76] Anders E. Zonst. *Understanding the FFT : a tutorial on the algorithm and software for laymen, students, technicians and working engineers.* Citrus Press, 1995.